# angr

The angr Project

Sep 17, 2026

# CONTENTS


**1 Introduction** **3**
- 1.1 Getting Support . . . 3
- 1.2 Citing angr . . . 3
- 1.3 Going further . . . 4

**2 Getting Started** **5**
- 2.1 Installing angr . . . 5
- 2.2 Reporting Bugs . . . 6
- 2.3 Developing angr . . . 7
- 2.4 Help Wanted . . . 9

**3 Core Concepts** **13**
- 3.1 Core Concepts . . . 13
- 3.2 Loading a Binary . . . 18
- 3.3 Symbolic Expressions and Constraint Solving . . . 24
- 3.4 Machine State - memory, registers, and so on . . . 30
- 3.5 Simulation Managers . . . 35
- 3.6 Simulation and Instrumentation . . . 39
- 3.7 Analyses . . . 46
- 3.8 Symbolic Execution . . . 47
- 3.9 A final word of advice . . . 48

**4 Build-in Analyses** **51**
- 4.1 Control-flow Graph Recovery (CFG) . . . 51
- 4.2 Backward Slicing . . . 55
- 4.3 Identifier . . . 58
- 4.4 angr Decompiler . . . 61

**5 Advanced Topics** **63**
- 5.1 Gotchas when using angr . . . 63
- 5.2 Understanding the Execution Pipeline . . . 64
- 5.3 What's Up With Mixins, Anyway? . . . 68
- 5.4 Optimization considerations . . . 71
- 5.5 Working with File System, Sockets, and Pipes . . . 73
- 5.6 Intermediate Representation . . . 78
- 5.7 Working with Data and Conventions . . . 81
- 5.8 Solver Engine . . . 85
- 5.9 Symbolic memory addressing . . . 89
- 5.10 Java Support . . . 90
- 5.11 Debug variable resolution . . . 91

5.12 Variable visibility . . . 92

**6 Extending angr** **95**
6.1 Hooks and SimProcedures . . . 95
6.2 State Plugins . . . 99
6.3 Extending the Environment Model . . . 102
6.4 Writing Analyses . . . 105

**7 angr examples** **109**
7.1 Introduction . . . 109
7.2 Reversing . . . 109
7.3 Vulnerability Discovery . . . 111
7.4 Exploitation . . . 112

**8 Frequently Asked Questions** **115**
8.1 Why is it named angr? . . . 115
8.2 How should "angr" be stylized? . . . 115
8.3 Why isn't symbolic execution doing the thing I want? . . . 115
8.4 How can I get diagnostic information about what angr is doing? . . . 115
8.5 Why is angr so slow? . . . 116
8.6 How do I find bugs using angr? . . . 116
8.7 Why did you choose VEX instead of another IR (such as LLVM, REIL, BAP, etc)? . . . 116
8.8 Why are some ARM addresses off-by-one? . . . 117
8.9 How do I serialize angr objects? . . . 117
8.10 What does `UnsupportedIROpError("floating point support disabled")` mean? . . . 117
8.11 Why is angr's CFG different from IDA's? . . . 118
8.12 Why do I get incorrect register values when reading from a state during a SimInspect breakpoint? . . . 118

**9 Appendix** **119**
9.1 Cheatsheet . . . 119
9.2 List of Claripy Operations . . . 123
9.3 List of State Options . . . 124
9.4 CTF Challenge Examples . . . 133
9.5 Changelog . . . 136
9.6 Migrating to angr 9.1 . . . 147
9.7 Migrating to angr 8 . . . 148
9.8 Migrating to angr 7 . . . 150

**10 API Reference** **155**
10.1 angr.ail_callable . . . 208
10.2 angr.ailment . . . 208
10.3 angr.analyses . . . 228
10.4 angr.angrdb . . . 938
10.5 angr.annocfg . . . 957
10.6 angr.blade . . . 958
10.7 angr.block . . . 959
10.8 angr.callable . . . 961
10.9 angr.calling_conventions . . . 962
10.10 angr.code_location . . . 985
10.11 angr.codenode . . . 987
10.12 angr.concretization_strategies . . . 988
10.13 angr.distributed . . . 993
10.14 angr.emulator . . . 995
10.15 angr.engines . . . 996
10.16 angr.errors . . . 1081

10.17 angr.exploration_techniques . . . 1087
10.18 angr.factory . . . 1109
10.19 angr.flirt . . . 1113
10.20 angr.keyed_region . . . 1115
10.21 angr.knowledge_base . . . 1118
10.22 angr.knowledge_plugins . . . 1120
10.23 angr.llm_client . . . 1293
10.24 angr.llm_models . . . 1294
10.25 angr.mcp . . . 1295
10.26 angr.misc . . . 1309
10.27 angr.procedures . . . 1322
10.28 angr.project . . . 1388
10.29 angr.protos . . . 1393
10.30 angr.rust . . . 1406
10.31 angr.rustylib . . . 1439
10.32 angr.serializable . . . 1440
10.33 angr.sim_manager . . . 1441
10.34 angr.sim_options . . . 1448
10.35 angr.sim_procedure . . . 1448
10.36 angr.sim_state . . . 1452
10.37 angr.sim_state_options . . . 1456
10.38 angr.sim_type . . . 1459
10.39 angr.sim_variable . . . 1477
10.40 angr.simos . . . 1479
10.41 angr.slicer . . . 1493
10.42 angr.state_hierarchy . . . 1494
10.43 angr.state_plugins . . . 1494
10.44 angr.storage . . . 1607
10.45 angr.tablespecs . . . 1668
10.46 angr.utils . . . 1668
10.47 angr.vaults . . . 1704

**11 Indices and tables** **1707**

**Python Module Index** **1709**

**Index** **1725**

Welcome to angr's documentation! This documentation is intended to be a guide for learning angr, as well as a reference for the API. If you're new to angr,

The angr team maintains a number of libraries that are used as part of angr. These libraries are:

- archinfo - Information about CPU architectures
- pyvex - Python bindings to the VEX IR
- pypcode - Python bindings to the Pcode IR
- cle - Many-platform binary loader
- claripy - Solver abstraction layer

angr also has a GUI! Check out angr-management.

CHAPTER

ONE

# INTRODUCTION

angr is a multi-architecture binary analysis toolkit, with the capability to perform dynamic symbolic execution (like Mayhem, KLEE, etc.) and various static analyses on binaries. If you'd like to learn how to use it, you're in the right place!

We've tried to make using angr as pain-free as possible - our goal is to create a user-friendly binary analysis suite, allowing a user to simply start up iPython and easily perform intensive binary analyses with a couple of commands. That being said, binary analysis is complex, which makes angr complex. This documentation is an attempt to help out with that, providing narrative explanation and exploration of angr and its design.

Several challenges must be overcome to programmatically analyze a binary. They are, roughly:

- Loading a binary into the analysis program.
- Translating a binary into an intermediate representation (IR).
- Performing the actual analysis. This could be:
  - A partial or full-program static analysis (i.e., dependency analysis, program slicing).
  - A symbolic exploration of the program's state space (i.e., "Can we execute it until we find an overflow?").
  - Some combination of the above (i.e., "Let's execute only program slices that lead to a memory write, to find an overflow.")

angr has components that meet all of these challenges. This documentation will explain how each component works, and how they can all be used to accomplish your goals.

## 1.1 Getting Support

To get help with angr, you can:

- Chat with us on the angr Discord server
- Open an issue on the appropriate GitHub repository

## 1.2 Citing angr

If you use angr in an academic work, please cite the papers for which it was developed:

```
@article{shoshitaishvili2016state,
  title={SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis},
  author={Shoshitaishvili, Yan and Wang, Ruoyu and Salls, Christopher and Stephens, Nick␣
↪and Polino, Mario and Dutcher, Audrey and Grosen, Jessie and Feng, Siji and Hauser,␣
↪Christophe and Kruegel, Christopher and Vigna, Giovanni},
  booktitle={IEEE Symposium on Security and Privacy},
```

(continues on next page)

(continued from previous page)

```
  year={2016}
}

@article{stephens2016driller,
  title={Driller: Augmenting Fuzzing Through Selective Symbolic Execution},
  author={Stephens, Nick and Grosen, Jessie and Salls, Christopher and Dutcher, Audrey␣
↪and Wang, Ruoyu and Corbetta, Jacopo and Shoshitaishvili, Yan and Kruegel, Christopher␣
↪and Vigna, Giovanni},
  booktitle={NDSS},
  year={2016}
}

@article{shoshitaishvili2015firmalice,
  title={Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in␣
↪Binary Firmware},
  author={Shoshitaishvili, Yan and Wang, Ruoyu and Hauser, Christophe and Kruegel,␣
↪Christopher and Vigna, Giovanni},
  booktitle={NDSS},
  year={2015}
}
```

## 1.3 Going further

You can read this paper, explaining some of the internals, algorithms, and used techniques to get a better understanding on what's going on under the hood.

If you enjoy playing CTFs and would like to learn angr in a similar fashion, angr_ctf will be a fun way for you to get familiar with much of the symbolic execution capability of angr. The angr_ctf repo is maintained by @jakespringer.

CHAPTER

TWO

# GETTING STARTED

## 2.1 Installing angr

angr is a library for Python 3.10+, and must be installed into a Python environment before it can be used.

### 2.1.1 Installing from PyPI

angr is published on PyPI, and using this is the easiest and recommended way to install angr. It can be installed angr with pip:

```
pip install angr
```

**Tip**

It is recommended to use an isolated python environment rather than installing angr globally. Doing so reduces dependency conflicts and aids in reproducibility while debugging. Some popular tools that accomplish this include:

- venv
- pipenv
- virtualenv
- virtualenvwrapper
- conda

**Note**

The PyPI distribution includes binary packages for most popular system configurations. If you are using a system that is not supported by the binary packages, you will need to build the C dependencies from source. See the *Installing from Source* section for more information.

### 2.1.2 Installing from Source

angr is a collection of Python packages, each of which is published on GitHub. The easiest way to install angr from source is to use angr-dev.

To set up a development environment manually, first ensure that build dependencies are installed. These consist of python development headers, `make`, a C++ compiler, and a Rust compiler. On Ubuntu, these can be installed with:

```
sudo apt-get install python3-dev build-essential
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

Then, checkout and install the following packages, in order:

- archinfo
- pyvex (clone with `--recursive`)
- cle
- claripy
- angr (`pip install` with `--no-build-isolation`)

### 2.1.3 Installing with Docker

The angr team maintains a container image on Docker Hub that includes angr and its dependencies. This image can be pulled with:

```
docker pull angr/angr
```

The image can be run with:

```
docker run -it angr/angr
```

This will start a shell in the container, with angr installed and ready to use.

### 2.1.4 Troubleshooting

#### angr has no attribute Project, or similar

If angr can be imported but the `Project` class is missing, it is likely one of two problems:

1. There is a script named `angr.py` in the working directory. Rename it to something else.
2. There is a folder called `angr` in your working directory, possibly the cloned repository. Change the working directory to somewhere else.

#### AttributeError: 'module' object has no attribute 'KS_ARCH_X86'

The `keystone` package is installed, which conflicts with the `keystone-engine` package, an optional dependency of angr. Uninstall `keystone` and install `keystone-engine`.

## 2.2 Reporting Bugs

If you've found something that angr isn't able to solve and appears to be a bug, please let us know!

1. Create a fork off of angr/binaries and angr/angr
2. Give us a pull request with angr/binaries, with the binaries in question
3. Give us a pull request for angr/angr, with testcases that trigger the binaries in `angr/tests/broken_x.py`, `angr/tests/broken_y.py`, etc

Please try to follow the testcase format that we have (so the code is in a test_blah function), that way we can very easily merge that and make the scripts run.

An example is:

```
def test_some_broken_feature():
    p = angr.Project("some_binary")
    result = p.analyses.SomethingThatDoesNotWork()
    assert result == "what it should *actually* be if it worked"

if __name__ == '__main__':
    test_some_broken_feature()
```

This will *greatly* help us recreate your bug and fix it faster.

The ideal situation is that, when the bug is fixed, your testcases passes (i.e., the assert at the end does not raise an AssertionError).

Then, we can just fix the bug and rename `broken_x.py` to `test_x.py` and the testcase will run in our internal CI at every push, ensuring that we do not break this feature again.

## 2.3 Developing angr

These are some guidelines so that we can keep the codebase in good shape!

### 2.3.1 pre-commit

Many angr repos contain pre-commit hooks provided by pre-commit. Installing this is as easy as `pip install pre-commit`. After `git` cloning an angr repository, if the repo contains a `.pre-commit-config.yaml`, run `pre-commit install`. Future `git` commits will now invoke these hooks automatically.

### 2.3.2 Coding style

We format our code with black and otherwise try to get as close as the PEP8 code convention as is reasonable without being dumb. If you use Vim, the python-mode plugin does all you need. You can also manually configure vim to adopt this behavior.

Most importantly, please consider the following when writing code as part of angr:

- Try to use attribute access (see the `@property` decorator) instead of getters and setters wherever you can. This isn't Java, and attributes enable tab completion in iPython. That being said, be reasonable: attributes should be fast. A rule of thumb is that if something could require a constraint solve, it should not be an attribute.
- Use our pylintrc from the angr-dev repo. It's fairly permissive, but our CI server will fail your builds if pylint complains under those settings.
- DO NOT, under ANY circumstances, `raise Exception` or `assert False`. **Use the right exception type**. If there isn't a correct exception type, subclass the core exception of the module that you're working in (i.e., `AngrError` in angr, `SimError` in SimuVEX, etc) and raise that. We catch, and properly handle, the right types of errors in the right places, but `AssertionError` and `Exception` are not handled anywhere and force-terminate analyses.
- Avoid tabs; use space indentation instead. Even though it's wrong, the de facto standard is 4 spaces. It is a good idea to adopt this from the beginning, as merging code that mixes both tab and space indentation is awful.
- Avoid super long lines. It's okay to have longer lines, but keep in mind that long lines are harder to read and should be avoided. Let's try to stick to **120 characters**.
- Avoid extremely long functions, it is often better to break them up into smaller functions.
- Always use _ instead of __ for private members (so that we can access them when debugging). *You* might not think that anyone has a need to call a given function, but trust us, you're wrong.

- Format your code with `black`; config is already defined within `pyproject.toml`.

### 2.3.3 Documentation

Document your code. Every *class definition* and *public function definition* should have some description of:

- What it does.
- What are the type and the meaning of the parameters.
- What it returns.

Class docstrings will be enforced by our linter. Do *not* under any circumstances write a docstring which doesn't provide more information than the name of the class. What you should try to write is a description of the environment that the class should be used in. If the class should not be instantiated by end-users, write a description of where it will be generated and how instances can be acquired. If the class should be instantiated by end-users, explain what kind of object it represents at its core, what behavior is expected of its parameters, and how to safely manage objects of its type.

We use Sphinx to generate the API documentation. Sphinx supports docstrings written in ReStructured Text with special keywords to document function and class parameters, return values, return types, members, etc.

Here is an example of function documentation. Ideally the parameter descriptions should be aligned vertically to make the docstrings as readable as possible.

```
def prune(self, filter_func=None, from_stash=None, to_stash=None):
    """
    Prune unsatisfiable paths from a stash.

    :param filter_func: Only prune paths that match this filter.
    :param from_stash:  Prune paths from this stash. (default: 'active')
    :param to_stash:    Put pruned paths in this stash. (default: 'pruned')
    :returns:           The resulting PathGroup.
    :rtype:             PathGroup
    """
```

This format has the advantage that the function parameters are clearly identified in the generated documentation. However, it can make the documentation repetitive, in some cases a textual description can be more readable. Pick the format you feel is more appropriate for the functions or classes you are documenting.

```
def read_bytes(self, addr, n):
   """
   Read `n` bytes at address `addr` in memory and return an array of bytes.
   """
```

### 2.3.4 Unit tests

If you're pushing a new feature and it is not accompanied by a test case it **will be broken** in very short order. Please write test cases for your stuff.

We have an internal CI server to run tests to check functionality and regression on each commit. In order to have our server run your tests, write your tests in a format acceptable to nosetests in a file matching `test_*.py` in the `tests` folder of the appropriate repository. A test file can contain any number of functions of the form `def test_*():` or classes of the form `class Test*(unittest.TestCase):`. Each of them will be run as a test, and if they raise any exceptions or assertions, the test fails. Do not use the `nose.tools.assert_*` functions, as we are presently trying to migrate to `nose2`. Use `assert` statements with descriptive messages or the `unittest.TestCase` assert methods.

Look at the existing tests for examples. Many of them use an alternate format where the `test_*` function is actually a generator that yields tuples of functions to call and their arguments, for easy parametrization of tests.

Finally, do not add docstrings to your test functions.

## 2.4 Help Wanted

**Todo**

This page is woefully out of date. We need to update it.

angr is a huge project, and it's hard to keep up. Here, we list some big TODO items that we would love community contributions for in the hope that it can direct community involvement. They (will) have a wide range of complexity, and there should be something for all skill levels!

We tag issues on our github repositories that would be good for community involvement as "Help wanted". To see the exhaustive list of these, use this github search!

### 2.4.1 Documentation

There are many parts of angr that suffer from little or no documentation. We desperately need community help in this area.

#### API

We are always behind on documentation. We've created several tracking issues on github to understand what's still missing:

1. angr
2. claripy
3. cle
4. pyvex

#### GitBook

This book is missing some core areas. Specifically, the following could be improved:

1. Finish some of the TODOs floating around the book.
2. Organize the Examples page in some way that makes sense. Right now, most of the examples are very redundant. It might be cool to have a simple table of most of them so that the page is not so overwhelming.

#### angr course

Developing a "course" of sorts to get people started with angr would be really beneficial. Steps have already been made in this direction here, but more expansion would be beneficial.

Ideally, the course would have a hands-on component, of increasing difficulty, that would require people to use more and more of angr's capabilities.

### 2.4.2 Research re-implementation

Unfortunately, not everyone bases their research on angr ;-). Until that's remedied, we'll need to periodically implement related work, on top of angr, to make it reusable within the scope of the framework. This section lists some of this related work that's ripe for reimplementation in angr.

#### Redundant State Detection for Dynamic Symbolic Execution

Bugrara, et al. describe a method to identify and trim redundant states, increasing the speed of symbolic execution by up to 50 times and coverage by 4%. This would be great to have in angr, as an ExplorationTechnique. The paper is here: http://nsl.cs.columbia.edu/projects/minestrone/papers/atc13-bugrara.pdf

#### In-Vivo Multi-Path Analysis of Software Systems

Rather than developing symbolic summaries for every system call, we can use a technique proposed by S2E for concretizing necessary data and dispatching them to the OS itself. This would make angr applicable to a *much* larger set of binaries than it can currently analyze.

While this would be most useful for system calls, once it is implemented, it could be trivially applied to any location of code (i.e., library functions). By carefully choosing which library functions are handled like this, we can greatly increase angr's scalability.

### 2.4.3 Development

We have several projects in mind that primarily require development effort.

#### angr-management

The angr GUI, angr-management, could use your improvements! Exposing more of angr's capabilities in a usable way, graphically, would be really useful!

#### IDA Plugins

Much of angr's functionality could be exposed via IDA. For example, angr's data dependence graph could be exposed in IDA through annotations, or obfuscated values can be resolved using symbolic execution.

#### Additional architectures

More architecture support would make angr all the more useful. Supporting a new architecture with angr would involve:

1. Adding the architecture information to archinfo
2. Adding an IR translation. This may be either an extension to PyVEX, producing IRSBs, or another IR entirely.
3. If your IR is not VEX, add a `SimEngine` to support it.
4. Adding a calling convention (`angr.SimCC`) to support SimProcedures (including system calls)
5. Adding or modifying an `angr.SimOS` to support initialization activities.
6. Creating a CLE backend to load binaries, or extending the CLE ELF backend to know about the new architecture if the binary format is ELF.

**ideas for new architectures:**

- PIC, AVR, other embedded architectures
- SPARC (there is some preliminary libVEX support for SPARC here)

**ideas for new IRs:**

- LLVM IR (with this, we can extend angr from just a Binary Analysis Framework to a Program Analysis Framework and expand its capabilities in other ways!)
- SOOT (there is no reason that angr can't analyze Java code, although doing so would require some extensions to our memory model)

#### Environment support

We use the concept of "function summaries" in angr to model the environment of operating systems (i.e., the effects of their system calls) and library functions. Extending this would be greatly helpful in increasing angr's utility. These function summaries can be found here.

A specific subset of this is system calls. Even more than library function SimProcedures (without which angr can always execute the actual function), we have very few workarounds for missing system calls. Every implemented system call extends the set of binaries that angr can handle.

### 2.4.4 Design Problems

There are some outstanding design challenges regarding the integration of additional functionalities into angr.

#### Type annotation and type information usage

angr has fledgling support for types, in the sense that it can parse them out of header files. However, those types are not well exposed to do anything useful with. Improving this support would make it possible to, for example, annotate certain memory regions with certain type information and interact with them intelligently. Consider, for example, interacting with a linked list like this: `print state.mem[state.regs.rax].llist.next.next.value`.

(editor's note: you can actually already do this)

### 2.4.5 Research Challenges

Historically, angr has progressed in the course of research into novel areas of program analysis. Here, we list several self-contained research projects that can be tackled.

#### Semantic function identification/diffing

Current function diffing techniques (TODO: some examples) have drawbacks. For the CGC, we created a semantic-based binary identification engine ( https://github.com/angr/identifier) that can identify functions based on testcases. There are two areas of improvement, each of which is its own research project:

1. Currently, the testcases used by this component are human-generated. However, symbolic execution can be used to automatically generate testcases that can be used to recognize instances of a given function in other binaries.
2. By creating testcases that achieve a "high-enough" code coverage of a given function, we can detect changes in functionality by applying the set of testcases to another implementation of the same function and analyzing changes in code coverage. This can then be used as a semantic function diff.

#### Applying AFL's path selection criteria to symbolic execution

AFL does an excellent job in identifying "unique" paths during fuzzing by tracking the control flow transitions taken by every path. This same metric can be applied to symbolic exploration, and would probably do a depressingly good job, considering how simple it is.

### 2.4.6 Overarching Research Directions

There are areas of program analysis that are not well explored. We list general directions of research here, but readers should keep in mind that these directions likely describe potential undertakings of entire PhD dissertations.

#### Process interactions

Almost all work in the field of binary analysis deals with single binaries, but this is often unrealistic in the real world. For example, the type of input that can be passed to a CGI program depend on pre-processing by a web server. Currently, there is no way to support the analysis of multiple concurrent processes in angr, and many open questions in the field (i.e., how to model concurrent actions).

#### Intra-process concurrency

Similar to the modeling of interactions between processes, little work has been done in understanding the interaction of concurrent threads in the same process. Currently, angr has no way to reason about this, and it is unclear from the theoretical perspective how to approach this.

A subset of this problem is the analysis of signal handlers (or hardware interrupts). Each signal handler can be modeled as a thread that can be executed at any time that a signal can be triggered. Understanding when it is meaningful to analyze these handlers is an open problem. One system that does reason about the effect of interrupts is FIE.

#### Path explosion

Many approaches (such as Veritesting) attempt to mitigate the path explosion problem in symbolic execution. However, despite these efforts, path explosion is still *the* main problem preventing symbolic execution from being mainstream.

angr provides an excellent base to implement new techniques to control path explosion. Most approaches can be easily implemented as `ExplorationTechnique` s and quickly evaluated (for example, on the CGC dataset).

CHAPTER

THREE

# CORE CONCEPTS

## 3.1 Core Concepts

To get started with angr, you'll need to have a basic overview of some fundamental angr concepts and how to construct some basic angr objects. We'll go over this by examining what's directly available to you after you've loaded a binary!

Your first action with angr will always be to load a binary into a *project*. We'll use `/bin/true` for these examples.

```
>>> import angr
>>> proj = angr.Project('/bin/true')
```

A project is your control base in angr. With it, you will be able to dispatch analyses and simulations on the executable you just loaded. Almost every single object you work with in angr will depend on the existence of a project in some form.

**Tip**

Using and exploring angr in IPython (or other Python command line interpreters) is a main use case that we design angr for. When you are not sure what interfaces are available, tab completion is your friend!

Sometimes tab completion in IPython can be slow. We find the following workaround helpful without degrading the validity of completion results:

```
# Drop this file in IPython profile's startup directory to avoid running it every time.
import IPython
py = IPython.get_ipython()
py.Completer.use_jedi = False
```

### 3.1.1 Basic properties

First, we have some basic properties about the project: its CPU architecture, its filename, and the address of its entry point.

```
>>> import monkeyhex # this will format numerical results in hexadecimal
>>> proj.arch
<Arch AMD64 (LE)>
>>> proj.entry
0x401670
>>> proj.filename
'/bin/true'
```

- *arch* is an instance of an `archinfo.Arch` object for whichever architecture the program is compiled, in this case little-endian amd64. It contains a ton of clerical data about the CPU it runs on, which you can peruse at your leisure. The common ones you care about are `arch.bits`, `arch.bytes` (that one is a `@property` declaration on the main Arch class), `arch.name`, and `arch.memory_endness`.
- *entry* is the entry point of the binary!
- *filename* is the absolute filename of the binary. Riveting stuff!

### 3.1.2 Loading

Getting from a binary file to its representation in a virtual address space is pretty complicated! We have a module called CLE to handle that. CLE's result, called the loader, is available in the `.loader` property. We'll get into detail on how to use this *soon*, but for now just know that you can use it to see the shared libraries that angr loaded alongside your program and perform basic queries about the loaded address space.

```
>>> proj.loader
<Loaded true, maps [0x400000:0x5004000]>

>>> proj.loader.shared_objects # may look a little different for you!
{'ld-linux-x86-64.so.2': <ELF Object ld-2.24.so, maps [0x2000000:0x2227167]>,
 'libc.so.6': <ELF Object libc-2.24.so, maps [0x1000000:0x13c699f]>}

>>> proj.loader.min_addr
0x400000
>>> proj.loader.max_addr
0x5004000

>>> proj.loader.main_object  # we've loaded several binaries into this project. Here's␣
↪the main one!
<ELF Object true, maps [0x400000:0x60721f]>

>>> proj.loader.main_object.execstack  # sample query: does this binary have an␣
↪executable stack?
False
>>> proj.loader.main_object.pic  # sample query: is this binary position-independent?
True
```

### 3.1.3 The factory

There are a lot of classes in angr, and most of them require a project to be instantiated. Instead of making you pass around the project everywhere, we provide `project.factory`, which has several convenient constructors for common objects you'll want to use frequently.

This section will also serve as an introduction to several basic angr concepts. Strap in!

#### Blocks

First, we have `project.factory.block()`, which is used to extract a basic block of code from a given address. This is an important fact - *angr analyzes code in units of basic blocks.* You will get back a Block object, which can tell you lots of fun things about the block of code:

```
>>> block = proj.factory.block(proj.entry) # lift a block of code from the program's␣
↪entry point
<Block for 0x401670, 42 bytes>
```

(continues on next page)

(continued from previous page)

```
>>> block.pp()                          # pretty-print a disassembly to stdout
0x401670:       xor     ebp, ebp
0x401672:       mov     r9, rdx
0x401675:       pop     rsi
0x401676:       mov     rdx, rsp
0x401679:       and     rsp, 0xfffffffffffffff0
0x40167d:       push    rax
0x40167e:       push    rsp
0x40167f:       lea     r8, [rip + 0x2e2a]
0x401686:       lea     rcx, [rip + 0x2db3]
0x40168d:       lea     rdi, [rip - 0xd4]
0x401694:       call    qword ptr [rip + 0x205866]

>>> block.instructions                  # how many instructions are there?
0xb
>>> block.instruction_addrs             # what are the addresses of the instructions?
[0x401670, 0x401672, 0x401675, 0x401676, 0x401679, 0x40167d, 0x40167e, 0x40167f,␣
↪0x401686, 0x40168d, 0x401694]
```

Additionally, you can use a Block object to get other representations of the block of code:

```
>>> block.capstone                       # capstone disassembly
<CapstoneBlock for 0x401670>
>>> block.vex                            # VEX IRSB (that's a Python internal address,␣
↪not a program address)
<pyvex.block.IRSB at 0x7706330>
```

### States

Here's another fact about angr - the `Project` object only represents an "initialization image" for the program. When you're performing execution with angr, you are working with a specific object representing a *simulated program state* - a `SimState`. Let's grab one right now!

```
>>> state = proj.factory.entry_state()
<SimState @ 0x401670>
```

A SimState contains a program's memory, registers, filesystem data... any "live data" that can be changed by execution has a home in the state. We'll cover how to interact with states in depth later, but for now, let's use `state.regs` and `state.mem` to access the registers and memory of this state:

```
>>> state.regs.rip        # get the current instruction pointer
<BV64 0x401670>
>>> state.regs.rax
<BV64 0x1c>
>>> state.mem[proj.entry].int.resolved  # interpret the memory at the entry point as a C␣
↪int
<BV32 0x8949ed31>
```

Those aren't Python ints! Those are *bitvectors*. Python integers don't have the same semantics as words on a CPU, e.g. wrapping on overflow, so we work with bitvectors, which you can think of as an integer as represented by a series of bits, to represent CPU data in angr. Note that each bitvector has a `.length` property describing how wide it is in bits.

We'll learn all about how to work with them soon, but for now, here's how to convert from Python ints to bitvectors and back again:

```
>>> bv = claripy.BVV(0x1234, 32)       # create a 32-bit-wide bitvector with value 0x1234
<BV32 0x1234>                              # BVV stands for bitvector value
>>> state.solver.eval(bv)                # convert to Python int
0x1234
```

You can store these bitvectors back to registers and memory, or you can directly store a Python integer and it'll be converted to a bitvector of the appropriate size:

```
>>> state.regs.rsi = claripy.BVV(3, 64)
>>> state.regs.rsi
<BV64 0x3>

>>> state.mem[0x1000].long = 4
>>> state.mem[0x1000].long.resolved
<BV64 0x4>
```

The `mem` interface is a little confusing at first, since it's using some pretty hefty Python magic. The short version of how to use it is:

- Use array[index] notation to specify an address
- Use `.<type>` to specify that the memory should be interpreted as `type` (common values: char, short, int, long, size_t, uint8_t, uint16_t...)
- From there, you can either:
  - Store a value to it, either a bitvector or a Python int
  - Use `.resolved` to get the value as a bitvector
  - Use `.concrete` to get the value as a Python int

There are more advanced usages that will be covered later!

Finally, if you try reading some more registers you may encounter a very strange looking value:

```
>>> state.regs.rdi
<BV64 reg_48_11_64{UNINITIALIZED}>
```

This is still a 64-bit bitvector, but it doesn't contain a numerical value. Instead, it has a name! This is called a *symbolic variable* and it is the underpinning of symbolic execution. Don't panic! We will discuss all of this in detail exactly two chapters from now.

### Simulation Managers

If a state lets us represent a program at a given point in time, there must be a way to get it to the *next* point in time. A simulation manager is the primary interface in angr for performing execution, simulation, whatever you want to call it, with states. As a brief introduction, let's show how to tick that state we created earlier forward a few basic blocks.

First, we create the simulation manager we're going to be using. The constructor can take a state or a list of states.

```
>>> simgr = proj.factory.simulation_manager(state)
<SimulationManager with 1 active>
>>> simgr.active
[<SimState @ 0x401670>]
```

A simulation manager can contain several *stashes* of states. The default stash, `active`, is initialized with the state we passed in. We could look at `simgr.active[0]` to look at our state some more, if we haven't had enough!

Now... get ready, we're going to do some execution.

```
>>> simgr.step()
```

We've just performed a basic block's worth of symbolic execution! We can look at the active stash again, noticing that it's been updated, and furthermore, that it has **not** modified our original state. SimState objects are treated as immutable by execution - you can safely use a single state as a "base" for multiple rounds of execution.

```
>>> simgr.active
[<SimState @ 0x1020300>]
>>> simgr.active[0].regs.rip                # new and exciting!
<BV64 0x1020300>
>>> state.regs.rip                          # still the same!
<BV64 0x401670>
```

`/bin/true` isn't a very good example for describing how to do interesting things with symbolic execution, so we'll stop here for now.

### 3.1.4 Analyses

angr comes pre-packaged with several built-in analyses that you can use to extract some fun kinds of information from a program. Here they are:

```
>>> proj.analyses.            # Press TAB here in ipython to get an autocomplete-listing␣
↪of everything:
 proj.analyses.BackwardSlice        proj.analyses.CongruencyCheck      proj.analyses.
↪reload_analyses
 proj.analyses.BinaryOptimizer      proj.analyses.DDG                  proj.analyses.
↪StaticHooker
 proj.analyses.BinDiff              proj.analyses.DFG                  proj.analyses.
↪VariableRecovery
 proj.analyses.BoyScout             proj.analyses.Disassembly          proj.analyses.
↪VariableRecoveryFast
 proj.analyses.CDG                  proj.analyses.GirlScout            proj.analyses.
↪Veritesting
 proj.analyses.CFG                  proj.analyses.Identifier           proj.analyses.VFG
 proj.analyses.CFGEmulated          proj.analyses.LoopFinder           proj.analyses.VSA_
↪DDG
 proj.analyses.CFGFast              proj.analyses.Reassembler
```

A couple of these are documented later in this book, but in general, if you want to find how to use a given analysis, you should look in the api documentation for *angr.analyses*. As an extremely brief example: here's how you construct and use a quick control-flow graph:

```
# Originally, when we loaded this binary it also loaded all its dependencies into the␣
↪same virtual address  space
# This is undesirable for most analysis.
>>> proj = angr.Project('/bin/true', auto_load_libs=False)
>>> cfg = proj.analyses.CFGFast()
<CFGFast Analysis Result at 0x2d85130>

# cfg.graph is a networkx DiGraph full of CFGNode instances
```

(continues on next page)

(continued from previous page)

```
# You should go look up the networkx APIs to learn how to use this!
>>> cfg.graph
<networkx.classes.digraph.DiGraph at 0x2da43a0>
>>> len(cfg.graph.nodes())
951

# To get the CFGNode for a given address, use cfg.model.get_any_node
>>> entry_node = cfg.model.get_any_node(proj.entry)
>>> len(list(cfg.graph.successors(entry_node)))
2
```

### 3.1.5 Now what?

Having read this page, you should now be acquainted with several important angr concepts: basic blocks, states, bitvectors, simulation managers, and analyses. You can't really do anything interesting besides just use angr as a glorified debugger, though! Keep reading, and you will unlock deeper powers...

## 3.2 Loading a Binary

Previously, you saw just the barest taste of angr's loading facilities - you loaded `/bin/true`, and then loaded it again without its shared libraries. You also saw `proj.loader` and a few things it could do. Now, we'll dive into the nuances of these interfaces and the things they can tell you.

We briefly mentioned angr's binary loading component, CLE. CLE stands for "CLE Loads Everything", and is responsible for taking a binary (and any libraries that it depends on) and presenting it to the rest of angr in a way that is easy to work with.

### 3.2.1 The Loader

Let's load `examples/fauxware/fauxware` and take a deeper look at how to interact with the loader.

```
>>> import angr, monkeyhex
>>> proj = angr.Project('examples/fauxware/fauxware')
>>> proj.loader
<Loaded fauxware, maps [0x400000:0x5008000]>
```

#### Loaded Objects

The CLE loader (`cle.Loader`) represents an entire conglomerate of loaded *binary objects*, loaded and mapped into a single memory space. Each binary object is loaded by a loader backend that can handle its filetype (a subclass of `cle.Backend`). For example, `cle.ELF` is used to load ELF binaries.

There will also be objects in memory that don't correspond to any loaded binary. For example, an object used to provide thread-local storage support, and an externs object used to provide unresolved symbols.

You can get the full list of objects that CLE has loaded with `loader.all_objects`, as well as several more targeted classifications:

```
# All loaded objects
>>> proj.loader.all_objects
[<ELF Object fauxware, maps [0x400000:0x60105f]>,
 <ELF Object libc-2.23.so, maps [0x1000000:0x13c999f]>,
 <ELF Object ld-2.23.so, maps [0x2000000:0x2227167]>,
```

(continues on next page)

(continued from previous page)

```
 <ELFTLSObject Object cle##tls, maps [0x3000000:0x3015010]>,
 <ExternObject Object cle##externs, maps [0x4000000:0x4008000]>,
 <KernelObject Object cle##kernel, maps [0x5000000:0x5008000]>]

# This is the "main" object, the one that you directly specified when loading the project
>>> proj.loader.main_object
<ELF Object fauxware, maps [0x400000:0x60105f]>

# This is a dictionary mapping from shared object name to object
>>> proj.loader.shared_objects
{ 'fauxware': <ELF Object fauxware, maps [0x400000:0x60105f]>,
  'libc.so.6': <ELF Object libc-2.23.so, maps [0x1000000:0x13c999f]>,
  'ld-linux-x86-64.so.2': <ELF Object ld-2.23.so, maps [0x2000000:0x2227167]> }

# Here's all the objects that were loaded from ELF files
# If this were a windows program we'd use all_pe_objects!
>>> proj.loader.all_elf_objects
[<ELF Object fauxware, maps [0x400000:0x60105f]>,
 <ELF Object libc-2.23.so, maps [0x1000000:0x13c999f]>,
 <ELF Object ld-2.23.so, maps [0x2000000:0x2227167]>]

# Here's the "externs object", which we use to provide addresses for unresolved imports␣
↪and angr internals
>>> proj.loader.extern_object
<ExternObject Object cle##externs, maps [0x4000000:0x4008000]>

# This object is used to provide addresses for emulated syscalls
>>> proj.loader.kernel_object
<KernelObject Object cle##kernel, maps [0x5000000:0x5008000]>

# Finally, you can to get a reference to an object given an address in it
>>> proj.loader.find_object_containing(0x400000)
<ELF Object fauxware, maps [0x400000:0x60105f]>
```

You can interact directly with these objects to extract metadata from them:

```
>>> obj = proj.loader.main_object

# The entry point of the object
>>> obj.entry
0x400580

>>> obj.min_addr, obj.max_addr
(0x400000, 0x60105f)

# Retrieve this ELF's segments and sections
>>> obj.segments
<Regions: [<ELFSegment memsize=0xa74, filesize=0xa74, vaddr=0x400000, flags=0x5,␣
↪offset=0x0>,
           <ELFSegment memsize=0x238, filesize=0x228, vaddr=0x600e28, flags=0x6,␣
↪offset=0xe28>]>
>>> obj.sections
```

(continues on next page)

(continued from previous page)

```
<Regions: [<Unnamed | offset 0x0, vaddr 0x0, size 0x0>,
           <.interp | offset 0x238, vaddr 0x400238, size 0x1c>,
           <.note.ABI-tag | offset 0x254, vaddr 0x400254, size 0x20>,
            ...etc

# You can get an individual segment or section by an address it contains:
>>> obj.find_segment_containing(obj.entry)
<ELFSegment memsize=0xa74, filesize=0xa74, vaddr=0x400000, flags=0x5, offset=0x0>
>>> obj.find_section_containing(obj.entry)
<.text | offset 0x580, vaddr 0x400580, size 0x338>

# Get the address of the PLT stub for a symbol
>>> addr = obj.plt['strcmp']
>>> addr
0x400550
>>> obj.reverse_plt[addr]
'strcmp'

# Show the prelinked base of the object and the location it was actually mapped into␣
↪memory by CLE
>>> obj.linked_base
0x400000
>>> obj.mapped_base
0x400000
```

### Symbols and Relocations

You can also work with symbols while using CLE. A symbol is a fundamental concept in the world of executable formats, effectively mapping a name to an address.

The easiest way to get a symbol from CLE is to use `loader.find_symbol`, which takes either a name or an address and returns a Symbol object.

```
>>> strcmp = proj.loader.find_symbol('strcmp')
>>> strcmp
<Symbol "strcmp" in libc.so.6 at 0x1089cd0>
```

The most useful attributes on a symbol are its name, its owner, and its address, but the "address" of a symbol can be ambiguous. The Symbol object has three ways of reporting its address:

- `.rebased_addr` is its address in the global address space. This is what is shown in the print output.
- `.linked_addr` is its address relative to the prelinked base of the binary. This is the address reported in, for example, `readelf(1)`.
- `.relative_addr` is its address relative to the object base. This is known in the literature (particularly the Windows literature) as an RVA (relative virtual address).

```
>>> strcmp.name
'strcmp'

>>> strcmp.owner
<ELF Object libc-2.23.so, maps [0x1000000:0x13c999f]>
```

(continues on next page)

(continued from previous page)

```
>>> strcmp.rebased_addr
0x1089cd0
>>> strcmp.linked_addr
0x89cd0
>>> strcmp.relative_addr
0x89cd0
```

In addition to providing debug information, symbols also support the notion of dynamic linking. libc provides the strcmp symbol as an export, and the main binary depends on it. If we ask CLE to give us a strcmp symbol from the main object directly, it'll tell us that this is an *import symbol*. Import symbols do not have meaningful addresses associated with them, but they do provide a reference to the symbol that was used to resolve them, as `.resolvedby`.

```
>>> strcmp.is_export
True
>>> strcmp.is_import
False

# On Loader, the method is find_symbol because it performs a search operation to find␣
↪the symbol.
# On an individual object, the method is get_symbol because there can only be one symbol␣
↪with a given name.
>>> main_strcmp = proj.loader.main_object.get_symbol('strcmp')
>>> main_strcmp
<Symbol "strcmp" in fauxware (import)>
>>> main_strcmp.is_export
False
>>> main_strcmp.is_import
True
>>> main_strcmp.resolvedby
<Symbol "strcmp" in libc.so.6 at 0x1089cd0>
```

The specific ways that the links between imports and exports should be registered in memory are handled by another notion called *relocations*. A relocation says, "when you match *[import]* up with an export symbol, please write the export's address to *[location]*, formatted as *[format]*." We can see the full list of relocations for an object (as `Relocation` instances) as `obj.relocs`, or just a mapping from symbol name to Relocation as `obj.imports`. There is no corresponding list of export symbols.

A relocation's corresponding import symbol can be accessed as `.symbol`. The address the relocation will write to is accessible through any of the address identifiers you can use for Symbol, and you can get a reference to the object requesting the relocation with `.owner` as well.

```
# Relocations don't have a good pretty-printing, so those addresses are Python-internal,␣
↪unrelated to our program
>>> proj.loader.shared_objects['libc.so.6'].imports
{'__libc_enable_secure': <cle.backends.elf.relocation.amd64.R_X86_64_GLOB_DAT at␣
↪0x7ff5c5fce780>,
 '__tls_get_addr': <cle.backends.elf.relocation.amd64.R_X86_64_JUMP_SLOT at␣
↪0x7ff5c6018358>,
 '_dl_argv': <cle.backends.elf.relocation.amd64.R_X86_64_GLOB_DAT at 0x7ff5c5fd2e48>,
 '_dl_find_dso_for_object': <cle.backends.elf.relocation.amd64.R_X86_64_JUMP_SLOT at␣
↪0x7ff5c6018588>,
 '_dl_starting_up': <cle.backends.elf.relocation.amd64.R_X86_64_GLOB_DAT at␣
↪0x7ff5c5fd2550>,
```

(continues on next page)

(continued from previous page)

```
 '_rtld_global': <cle.backends.elf.relocation.amd64.R_X86_64_GLOB_DAT at 0x7ff5c5fce4e0>,
 '_rtld_global_ro': <cle.backends.elf.relocation.amd64.R_X86_64_GLOB_DAT at␣
↪0x7ff5c5fcea20>}
```

If an import cannot be resolved to any export, for example, because a shared library could not be found, CLE will automatically update the externs object (`loader.extern_obj`) to claim it provides the symbol as an export.

### 3.2.2 Loading Options

If you are loading something with `angr.Project` and you want to pass an option to the `cle.Loader` instance that Project implicitly creates, you can just pass the keyword argument directly to the Project constructor, and it will be passed on to CLE. You should look at the CLE API docs. if you want to know everything that could possibly be passed in as an option, but we will go over some important and frequently used options here.

#### Basic Options

We've discussed `auto_load_libs` already - it enables or disables CLE's attempt to automatically resolve shared library dependencies, and is off by default. Additionally, there is the opposite, `except_missing_libs`, which, if set to true, will cause an exception to be thrown whenever a binary has a shared library dependency that cannot be resolved.

You can pass a list of strings to `force_load_libs` and anything listed will be treated as an unresolved shared library dependency right out of the gate, or you can pass a list of strings to `skip_libs` to prevent any library of that name from being resolved as a dependency. Additionally, you can pass a list of strings (or a single string) to `ld_path`, which will be used as an additional search path for shared libraries, before any of the defaults: the same directory as the loaded program, the current working directory, and your system libraries.

#### Per-Binary Options

If you want to specify some options that only apply to a specific binary object, CLE will let you do that too. The parameters `main_opts` and `lib_opts` do this by taking dictionaries of options. `main_opts` is a mapping from option names to option values, while `lib_opts` is a mapping from library name to dictionaries mapping option names to option values.

The options that you can use vary from backend to backend, but some common ones are:

- `backend` - which backend to use, as either a class or a name
- `base_addr` - a base address to use
- `entry_point` - an entry point to use
- `arch` - the name of an architecture to use

Example:

```
>>> angr.Project('examples/fauxware/fauxware', main_opts={'backend': 'blob', 'arch':
↪'i386'}, lib_opts={'libc.so.6': {'backend': 'elf'}})
<Project examples/fauxware/fauxware>
```

#### Backends

CLE currently has backends for statically loading ELF, PE, CGC, Mach-O and ELF core dump files, as well as loading files into a flat address space. CLE will automatically detect the correct backend to use in most cases, so you shouldn't need to specify which backend you're using unless you're doing some pretty weird stuff.

You can force CLE to use a specific backend for an object by including a key in its options dictionary, as described above. Some backends cannot autodetect which architecture to use and *must* have a `arch` specified. The key doesn't

need to match any list of architectures; angr will identify which architecture you mean given almost any common identifier for any supported arch.

To refer to a backend, use the name from this table:

| backend name | description | requires `arch`? |
|---|---|---|
| elf | Static loader for ELF files based on PyELFTools | no |
| pe | Static loader for PE files based on PEFile | no |
| mach-o | Static loader for Mach-O files. Does not support dynamic linking or rebasing. | no |
| cgc | Static loader for Cyber Grand Challenge binaries | no |
| backedcgc | Static loader for CGC binaries that allows specifying memory and register backers | no |
| elfcore | Static loader for ELF core dumps | no |
| blob | Loads the file into memory as a flat image | yes |

### 3.2.3 Symbolic Function Summaries

By default, Project tries to replace external calls to library functions by using symbolic summaries termed *SimProcedures* - effectively just Python functions that imitate the library function's effect on the state. We've implemented a whole bunch of functions as SimProcedures. These builtin procedures are available in the `angr.SIM_PROCEDURES` dictionary, which is two-leveled, keyed first on the package name (libc, posix, win32, stubs) and then on the name of the library function. Executing a SimProcedure instead of the actual library function that gets loaded from your system makes analysis a LOT more tractable, at the cost of some potential inaccuracies.

When no such summary is available for a given function:

- if `auto_load_libs` is `True`, then the *real* library function is executed instead. This may or may not be what you want, depending on the actual function. For example, some of libc's functions are extremely complex to analyze and will most likely cause an explosion of the number of states for the path trying to execute them.
- if `auto_load_libs` is `False` (this is the default), then external functions are unresolved, and Project will resolve them to a generic "stub" SimProcedure called `ReturnUnconstrained`. It does what its name says: it returns a unique unconstrained symbolic value each time it is called.
- if `use_sim_procedures` (this is a parameter to `angr.Project`, not `cle.Loader`) is `False` (it is `True` by default), then only symbols provided by the extern object will be replaced with SimProcedures, and they will be replaced by a stub `ReturnUnconstrained`, which does nothing but return a symbolic value.
- you may specify specific symbols to exclude from being replaced with SimProcedures with the parameters to `angr.Project`: `exclude_sim_procedures_list` and `exclude_sim_procedures_func`.
- Look at the code for `angr.Project._register_object` for the exact algorithm.

#### Hooking

The mechanism by which angr replaces library code with a Python summary is called hooking, and you can do it too! When performing simulation, at every step angr checks if the current address has been hooked, and if so, runs the hook instead of the binary code at that address. The API to let you do this is `proj.hook(addr, hook)`, where `hook` is a SimProcedure instance. You can manage your project's hooks with `.is_hooked`, `.unhook`, and `.hooked_by`, which should hopefully not require explanation.

There is an alternate API for hooking an address that lets you specify your own off-the-cuff function to use as a hook, by using `proj.hook(addr)` as a function decorator. If you do this, you can also optionally specify a `length` keyword argument to make execution jump some number of bytes forward after your hook finishes.

```
>>> stub_func = angr.SIM_PROCEDURES['stubs']['ReturnUnconstrained'] # this is a CLASS
>>> proj.hook(0x10000, stub_func())  # hook with an instance of the class

>>> proj.is_hooked(0x10000)            # these functions should be pretty self-
↪explanitory
True
>>> proj.hooked_by(0x10000)
<ReturnUnconstrained>
>>> proj.unhook(0x10000)

>>> @proj.hook(0x20000, length=5)
... def my_hook(state):
...     state.regs.rax = 1

>>> proj.is_hooked(0x20000)
True
```

Furthermore, you can use `proj.hook_symbol(name, hook)`, providing the name of a symbol as the first argument, to hook the address where the symbol lives. One very important usage of this is to extend the behavior of angr's built-in library SimProcedures. Since these library functions are just classes, you can subclass them, overriding pieces of their behavior, and then use your subclass in a hook.

### 3.2.4 So far so good!

By now, you should have a reasonable understanding of how to control the environment in which your analysis happens, on the level of the CLE loader and the angr Project. You should also understand that angr makes a reasonable attempt to simplify its analysis by hooking complex library functions with SimProcedures that summarize the effects of the functions.

In order to see all the things you can do with the CLE loader and its backends, look at the CLE API docs.

## 3.3 Symbolic Expressions and Constraint Solving

angr's power comes not from it being an emulator, but from being able to execute with what we call *symbolic variables*. Instead of saying that a variable has a *concrete* numerical value, we can say that it holds a *symbol*, effectively just a name. Then, performing arithmetic operations with that variable will yield a tree of operations (termed an *abstract syntax tree* or *AST*, from compiler theory). ASTs can be translated into constraints for an *SMT solver*, like z3, in order to ask questions like *"given the output of this sequence of operations, what must the input have been?"* Here, you'll learn how to use angr to answer this.

### 3.3.1 Working with Bitvectors

Let's get a dummy project and state so we can start playing with numbers.

```
>>> import angr, monkeyhex
>>> proj = angr.Project('/bin/true')
>>> state = proj.factory.entry_state()
```

A bitvector is just a sequence of bits, interpreted with the semantics of a bounded integer for arithmetic. Let's make a few.

```
# 64-bit bitvectors with concrete values 1 and 100
>>> one = claripy.BVV(1, 64)
```

(continues on next page)

(continued from previous page)

```
>>> one
 <BV64 0x1>
>>> one_hundred =claripy.BVV(100, 64)
>>> one_hundred
 <BV64 0x64>

# create a 27-bit bitvector with concrete value 9
>>> weird_nine = claripy.BVV(9, 27)
>>> weird_nine
<BV27 0x9>
```

As you can see, you can have any sequence of bits and call them a bitvector. You can do math with them too:

```
>>> one + one_hundred
<BV64 0x65>

# You can provide normal Python integers and they will be coerced to the
appropriate type: >>> one_hundred + 0x100 <BV64 0x164>

# The semantics of normal wrapping arithmetic apply
>>> one_hundred - one*200
<BV64 0xffffffffffffff9c>
```

You *cannot* say `one + weird_nine`, though. It is a type error to perform an operation on bitvectors of differing lengths. You can, however, extend `weird_nine` so it has an appropriate number of bits:

```
>>> weird_nine.zero_extend(64 - 27)
<BV64 0x9>
>>> one + weird_nine.zero_extend(64 - 27)
<BV64 0xa>
```

`zero_extend` will pad the bitvector on the left with the given number of zero bits. You can also use `sign_extend` to pad with a duplicate of the highest bit, preserving the value of the bitvector under two's complement signed integer semantics.

Now, let's introduce some symbols into the mix.

```
# Create a bitvector symbol named "x" of length 64 bits
>>> x = claripy.BVS("x", 64)
>>> x
<BV64 x_9_64>
>>> y = claripy.BVS("y", 64)
>>> y
<BV64 y_10_64>
```

`x` and `y` are now *symbolic variables*, which are kind of like the variables you learned to work with in 7th grade algebra. Notice that the name you provided has been mangled by appending an incrementing counter and You can do as much arithmetic as you want with them, but you won't get a number back, you'll get an AST instead.

```
>>> x + one
<BV64 x_9_64 + 0x1>

>>> (x + one) / 2
```

(continues on next page)

(continued from previous page)

```
<BV64 (x_9_64 + 0x1) / 0x2>

>>> x - y
<BV64 x_9_64 - y_10_64>
```

Technically `x` and `y` and even `one` are also ASTs - any bitvector is a tree of operations, even if that tree is only one layer deep. To understand this, let's learn how to process ASTs.

Each AST has a `.op` and a `.args`. The op is a string naming the operation being performed, and the args are the values the operation takes as input. Unless the op is `BVV` or `BVS` (or a few others...), the args are all other ASTs, the tree eventually terminating with BVVs or BVSs.

```
>>> tree = (x + 1) / (y + 2)
>>> tree
<BV64 (x_9_64 + 0x1) / (y_10_64 + 0x2)>
>>> tree.op
'__floordiv__'
>>> tree.args
(<BV64 x_9_64 + 0x1>, <BV64 y_10_64 + 0x2>)
>>> tree.args[0].op
'__add__'
>>> tree.args[0].args
(<BV64 x_9_64>, <BV64 0x1>)
>>> tree.args[0].args[1].op
'BVV'
>>> tree.args[0].args[1].args
(1, 64)
```

From here on out, we will use the word "bitvector" to refer to any AST whose topmost operation produces a bitvector. There can be other data types represented through ASTs, including floating point numbers and, as we're about to see, booleans.

### 3.3.2 Symbolic Constraints

Performing comparison operations between any two similarly-typed ASTs will yield another AST - not a bitvector, but now a symbolic boolean.

```
>>> x == 1
<Bool x_9_64 == 0x1>
>>> x == one
<Bool x_9_64 == 0x1>
>>> x > 2
<Bool x_9_64 > 0x2>
>>> x + y == one_hundred + 5
<Bool (x_9_64 + y_10_64) == 0x69>
>>> one_hundred > 5
<Bool True>
>>> one_hundred > -5
<Bool False>
```

One tidbit you can see from this is that the comparisons are unsigned by default. The -5 in the last example is coerced to `<BV64 0xfffffffffffffffb>`, which is definitely not less than one hundred. If you want the comparison to be signed, you can say `one_hundred.SGT(-5)` (that's "signed greater-than"). A full list of operations can be found at the end of this chapter.

This snippet also illustrates an important point about working with angr - you should never directly use a comparison between variables in the condition for an if- or while-statement, since the answer might not have a concrete truth value. Even if there is a concrete truth value, `if one > one_hundred` will raise an exception. Instead, you should use `solver.is_true` and `solver.is_false`, which test for concrete truthyness/falsiness without performing a constraint solve.

```
>>> yes = one == 1
>>> no = one == 2
>>> maybe = x == y
>>> state.solver.is_true(yes)
True
>>> state.solver.is_false(yes)
False
>>> state.solver.is_true(no)
False
>>> state.solver.is_false(no)
True
>>> state.solver.is_true(maybe)
False
>>> state.solver.is_false(maybe)
False
```

### 3.3.3 Constraint Solving

You can treat any symbolic boolean as an assertion about the valid values of a symbolic variable by adding it as a *constraint* to the state. You can then query for a valid value of a symbolic variable by asking for an evaluation of a symbolic expression.

An example will probably be more clear than an explanation here:

```
>>> state.solver.add(x > y)
>>> state.solver.add(y > 2)
>>> state.solver.add(10 > x)
>>> state.solver.eval(x)
4
```

By adding these constraints to the state, we've forced the constraint solver to consider them as assertions that must be satisfied about any values it returns. If you run this code, you might get a different value for x, but that value will definitely be greater than 3 (since y must be greater than 2 and x must be greater than y) and less than 10. Furthermore, if you then say `state.solver.eval(y)`, you'll get a value of y which is consistent with the value of x that you got. If you don't add any constraints between two queries, the results will be consistent with each other.

From here, it's easy to see how to do the task we proposed at the beginning of the chapter - finding the input that produced a given output.

```
# get a fresh state without constraints
>>> state = proj.factory.entry_state()
>>> input = claripy.BVS('input', 64)
>>> operation = (((input + 4) * 3) >> 1) + input
>>> output = 200
>>> state.solver.add(operation == output)
>>> state.solver.eval(input)
0x3333333333333381
```

Note that, again, this solution only works because of the bitvector semantics. If we were operating over the domain of

integers, there would be no solutions!

If we add conflicting or contradictory constraints, such that there are no values that can be assigned to the variables such that the constraints are satisfied, the state becomes *unsatisfiable*, or unsat, and queries against it will raise an exception. You can check the satisfiability of a state with `state.satisfiable()`.

```
>>> state.solver.add(input < 2**32)
>>> state.satisfiable()
False
```

You can also evaluate more complex expressions, not just single variables.

```
# fresh state
>>> state = proj.factory.entry_state()
>>> state.solver.add(x - y >= 4)
>>> state.solver.add(y > 0)
>>> state.solver.eval(x)
5
>>> state.solver.eval(y)
1
>>> state.solver.eval(x + y)
6
```

From this we can see that `eval` is a general purpose method to convert any bitvector into a Python primitive while respecting the integrity of the state. This is why we use `eval` to convert from concrete bitvectors to Python ints, too!

Also note that the x and y variables can be used in this new state despite having been created using an old state. Variables are not tied to any one state, and can exist freely.

### 3.3.4 Floating point numbers

z3 has support for the theory of IEEE754 floating point numbers, and so angr can use them as well. The main difference is that instead of a width, a floating point number has a *sort*. You can create floating point symbols and values with `FPV` and `FPS`.

```
# fresh state
>>> state = proj.factory.entry_state()
>>> a = claripy.FPV(3.2, claripy.fp.FSORT_DOUBLE)
>>> a
<FP64 FPV(3.2, DOUBLE)>

>>> b = claripy.FPS('b', claripy.fp.FSORT_DOUBLE)
>>> b
<FP64 FPS('FP_b_0_64', DOUBLE)>

>>> a + b
<FP64 fpAdd('RNE', FPV(3.2, DOUBLE), FPS('FP_b_0_64', DOUBLE))>

>>> a + 4.4
<FP64 FPV(7.6000000000000005, DOUBLE)>

>>> b + 2 < 0
<Bool fpLT(fpAdd('RNE', FPS('FP_b_0_64', DOUBLE), FPV(2.0, DOUBLE)), FPV(0.0, DOUBLE))>
```

So there's a bit to unpack here - for starters the pretty-printing isn't as smart about floating point numbers. But past that, most operations actually have a third parameter, implicitly added when you use the binary operators - the rounding

mode. The IEEE754 spec supports multiple rounding modes (round-to-nearest, round-to-zero, round-to-positive, etc), so z3 has to support them. If you want to specify the rounding mode for an operation, use the fp operation explicitly (`claripy.fpAdd` for example) with a rounding mode (one of `claripy.fp.RM_*`) as the first argument.

Constraints and solving work in the same way, but with `eval` returning a floating point number:

```
>>> state.solver.add(b + 2 < 0)
>>> state.solver.add(b + 2 > -1)
>>> state.solver.eval(b)
-2.4999999999999996
```

This is nice, but sometimes we need to be able to work directly with the representation of the float as a bitvector. You can interpret bitvectors as floats and vice versa, with the methods `raw_to_bv` and `raw_to_fp`:

```
>>> a.raw_to_bv()
<BV64 0x400999999999999a>
>>> b.raw_to_bv()
<BV64 fpToIEEEBV(FPS('FP_b_0_64', DOUBLE))>

>>> claripy.BVV(0, 64).raw_to_fp()
<FP64 FPV(0.0, DOUBLE)>
>>> claripy.BVS('x', 64).raw_to_fp()
<FP64 fpToFP(x_1_64, DOUBLE)>
```

These conversions preserve the bit-pattern, as if you casted a float pointer to an int pointer or vice versa. However, if you want to preserve the value as closely as possible, as if you casted a float to an int (or vice versa), you can use a different set of methods, `val_to_fp` and `val_to_bv`. These methods must take the size or sort of the target value as a parameter, due to the floating-point nature of floats.

```
>>> a
<FP64 FPV(3.2, DOUBLE)>
>>> a.val_to_bv(12)
<BV12 0x3>
>>> a.val_to_bv(12).val_to_fp(claripy.fp.FSORT_FLOAT)
<FP32 FPV(3.0, FLOAT)>
```

These methods can also take a `signed` parameter, designating the signedness of the source or target bitvector.

### 3.3.5 More Solving Methods

`eval` will give you one possible solution to an expression, but what if you want several? What if you want to ensure that the solution is unique? The solver provides you with several methods for common solving patterns:

- `solver.eval(expression)` will give you one possible solution to the given expression.
- `solver.eval_one(expression)` will give you the solution to the given expression, or throw an error if more than one solution is possible.
- `solver.eval_upto(expression, n)` will give you up to n solutions to the given expression, returning fewer than n if fewer than n are possible.
- `solver.eval_atleast(expression, n)` will give you n solutions to the given expression, throwing an error if fewer than n are possible.
- `solver.eval_exact(expression, n)` will give you n solutions to the given expression, throwing an error if fewer or more than are possible.
- `solver.min(expression)` will give you the minimum possible solution to the given expression.

- `solver.max(expression)` will give you the maximum possible solution to the given expression.

Additionally, all of these methods can take the following keyword arguments:

- `extra_constraints` can be passed as a tuple of constraints. These constraints will be taken into account for this evaluation, but will not be added to the state.
- `cast_to` can be passed a data type to cast the result to. Currently, this can only be `int` and `bytes`, which will cause the method to return the corresponding representation of the underlying data. For example, `state.solver.eval(claripy.BVV(0x41424344, 32), cast_to=bytes)` will return `b'ABCD'`.

### 3.3.6 Summary

That was a lot!! After reading this, you should be able to create and manipulate bitvectors, booleans, and floating point values to form trees of operations, and then query the constraint solver attached to a state for possible solutions under a set of constraints. Hopefully by this point you understand the power of using ASTs to represent computations, and the power of a constraint solver.

In the appendix, you can find a reference for all the additional operations you can apply to ASTs, in case you ever need a quick table to look at.

## 3.4 Machine State - memory, registers, and so on

So far, we've only used angr's simulated program states (`SimState` objects) in the barest possible way in order to demonstrate basic concepts about angr's operation. Here, you'll learn about the structure of a state object and how to interact with it in a variety of useful ways.

### 3.4.1 Review: Reading and writing memory and registers

If you've been reading this book in order (and you should be, at least for this first section), you already saw the basics of how to access memory and registers. `state.regs` provides read and write access to the registers through attributes with the names of each register, and `state.mem` provides typed read and write access to memory with index-access notation to specify the address followed by an attribute access to specify the type you would like to interpret the memory as.

Additionally, you should now know how to work with ASTs, so you can now understand that any bitvector-typed AST can be stored in registers or memory.

Here are some quick examples for copying and performing operations on data from the state:

```
>>> import angr, claripy
>>> proj = angr.Project('/bin/true')
>>> state = proj.factory.entry_state()

# copy rsp to rbp
>>> state.regs.rbp = state.regs.rsp

# store rdx to memory at 0x1000
>>> state.mem[0x1000].uint64_t = state.regs.rdx

# dereference rbp
>>> state.regs.rbp = state.mem[state.regs.rbp].uint64_t.resolved

# add rax, qword ptr [rsp + 8]
>>> state.regs.rax += state.mem[state.regs.rsp + 8].uint64_t.resolved
```

### 3.4.2 Basic Execution

Earlier, we showed how to use a Simulation Manager to do some basic execution. We'll show off the full capabilities of the simulation manager in the next chapter, but for now we can use a much simpler interface to demonstrate how symbolic execution works: `state.step()`. This method will perform one step of symbolic execution and return an object called *angr.engines.successors.SimSuccessors*. Unlike normal emulation, symbolic execution can produce several successor states that can be classified in a number of ways. For now, what we care about is the `.successors` property of this object, which is a list containing all the "normal" successors of a given step.

Why a list, instead of just a single successor state? Well, angr's process of symbolic execution is just the taking the operations of the individual instructions compiled into the program and performing them to mutate a SimState. When a line of code like `if (x > 4)` is reached, what happens if x is a symbolic bitvector? Somewhere in the depths of angr, the comparison `x > 4` is going to get performed, and the result is going to be `<Bool x_32_1 > 4>`.

That's fine, but the next question is, do we take the "true" branch or the "false" one? The answer is, we take both! We generate two entirely separate successor states - one simulating the case where the condition was true and simulating the case where the condition was false. In the first state, we add `x > 4` as a constraint, and in the second state, we add `!(x > 4)` as a constraint. That way, whenever we perform a constraint solve using either of these successor states, *the conditions on the state ensure that any solutions we get are valid inputs that will cause execution to follow the same path that the given state has followed.*

To demonstrate this, let's use a *fake firmware image <../examples/fauxware/fauxware>* as an example. If you look at the *source code <../examples/fauxware/fauxware.c>* for this binary, you'll see that the authentication mechanism for the firmware is backdoored; any username can be authenticated as an administrator with the password "SOSNEAKY". Furthermore, the first comparison against user input that happens is the comparison against the backdoor, so if we step until we get more than one successor state, one of those states will contain conditions constraining the user input to be the backdoor password. The following snippet implements this:

```
>>> proj = angr.Project('examples/fauxware/fauxware')
>>> state = proj.factory.entry_state(stdin=angr.SimFile)  # ignore that argument for now␣
↪- we're disabling a more complicated default setup for the sake of education
>>> while True:
...     succ = state.step()
...     if len(succ.successors) == 2:
...         break
...     state = succ.successors[0]

>>> state1, state2 = succ.successors
>>> state1
<SimState @ 0x400629>
>>> state2
<SimState @ 0x400699
```

Don't look at the constraints on these states directly - the branch we just went through involves the result of `strcmp`, which is a tricky function to emulate symbolically, and the resulting constraints are *very* complicated.

The program we emulated took data from standard input, which angr treats as an infinite stream of symbolic data by default. To perform a constraint solve and get a possible value that input could have taken in order to satisfy the constraints, we'll need to get a reference to the actual contents of stdin. We'll go over how our file and input subsystems work later on this very page, but for now, just use `state.posix.stdin.load(0, state.posix.stdin.size)` to retrieve a bitvector representing all the content read from stdin so far.

```
>>> input_data = state1.posix.stdin.load(0, state1.posix.stdin.size)

>>> state1.solver.eval(input_data, cast_to=bytes)
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00SOSNEAKY\x00\x00\x00'
```

(continues on next page)

(continued from previous page)

```
>>> state2.solver.eval(input_data, cast_to=bytes)
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00S\x00\x80N\x00\x00 \x00\x00\x00\x00'
```

As you can see, in order to go down the `state1` path, you must have given as a password the backdoor string "SOSNEAKY". In order to go down the `state2` path, you must have given something *besides* "SOSNEAKY". z3 has helpfully provided one of the billions of strings fitting this criteria.

Fauxware was the first program angr's symbolic execution ever successfully worked on, back in 2013. By finding its backdoor using angr you are participating in a grand tradition of having a bare-bones understanding of how to use symbolic execution to extract meaning from binaries!

### 3.4.3 State Presets

So far, whenever we've been working with a state, we've created it with `project.factory.entry_state()`. This is just one of several *state constructors* available on the project factory:

- `.blank_state()` constructs a "blank slate" blank state, with most of its data left uninitialized. When accessing uninitialized data, an unconstrained symbolic value will be returned.
- `.entry_state()` constructs a state ready to execute at the main binary's entry point.
- `.full_init_state()` constructs a state that is ready to execute through any initializers that need to be run before the main binary's entry point, for example, shared library constructors or preinitializers. When it is finished with these it will jump to the entry point.
- `.call_state()` constructs a state ready to execute a given function.

You can customize the state through several arguments to these constructors:

- All of these constructors can take an `addr` argument to specify the exact address to start.
- If you're executing in an environment that can take command line arguments or an environment, you can pass a list of arguments through `args` and a dictionary of environment variables through `env` into `entry_state` and `full_init_state`. The values in these structures can be strings or bitvectors, and will be serialized into the state as the arguments and environment to the simulated execution. The default `args` is an empty list, so if the program you're analyzing expects to find at least an `argv[0]`, you should always provide that!
- If you'd like to have `argc` be symbolic, you can pass a symbolic bitvector as `argc` to the `entry_state` and `full_init_state` constructors. Be careful, though: if you do this, you should also add a constraint to the resulting state that your value for argc cannot be larger than the number of args you passed into `args`.
- To use the call state, you should call it with `.call_state(addr, arg1, arg2, ...)`, where `addr` is the address of the function you want to call and `argN` is the Nth argument to that function, either as a Python integer, string, or array, or a bitvector. If you want to have memory allocated and actually pass in a pointer to an object, you should wrap it in an PointerWrapper, i.e. `angr.PointerWrapper("point to me!")`. The results of this API can be a little unpredictable, but we're working on it.
- To specify the calling convention used for a function with `call_state`, you can pass a *SimCC* instance as the `cc` argument. We try to pick a sane default, but for special cases you will need to help angr out.

There are several more options that can be used in any of these constructors! See the docs on the `project.factory` object (an *angr.factory.AngrObjectFactory*) for more details.

### 3.4.4 Low level interface for memory

The `state.mem` interface is convenient for loading typed data from memory, but when you want to do raw loads and stores to and from ranges of memory, it's very cumbersome. It turns out that `state.mem` is actually just a bunch of logic to correctly access the underlying memory storage, which is just a flat address space filled with bitvector data: `state.memory`. You can use `state.memory` directly with the `.load(addr, size)` and `.store(addr, val)` methods:

```
>>> s = proj.factory.blank_state()
>>> s.memory.store(0x4000, claripy.BVV(0x0123456789abcdef0123456789abcdef, 128))
>>> s.memory.load(0x4004, 6) # load-size is in bytes
<BV48 0x89abcdef0123>
```

As you can see, the data is loaded and stored in a "big-endian" fashion, since the primary purpose of `state.memory` is to load an store swaths of data with no attached semantics. However, if you want to perform a byteswap on the loaded or stored data, you can pass a keyword argument `endness` - if you specify little-endian, byteswap will happen. The endness should be one of the members of the `Endness` enum in the `archinfo` package used to hold declarative data about CPU architectures for angr. Additionally, the endness of the program being analyzed can be found as `arch.memory_endness` - for instance `state.arch.memory_endness`.

```
>>> import archinfo
>>> s.memory.load(0x4000, 4, endness=archinfo.Endness.LE)
<BV32 0x67452301>
```

There is also a low-level interface for register access, `state.registers`, that uses the exact same API as `state.memory`, but explaining its behavior involves a *dive* into the abstractions that angr uses to seamlessly work with multiple architectures. The short version is that it is simply a register file, with the mapping between registers and offsets defined in archinfo.

### 3.4.5 State Options

There are a lot of little tweaks that can be made to the internals of angr that will optimize behavior in some situations and be a detriment in others. These tweaks are controlled through state options.

On each SimState object, there is a set (`state.options`) of all its enabled options. Each option (really just a string) controls the behavior of angr's execution engine in some minute way. A listing of the full domain of options, along with the defaults for different state types, can be found in *the appendix*. You can access an individual option for adding to a state through `angr.options`. The individual options are named with CAPITAL_LETTERS, but there are also common groupings of objects that you might want to use bundled together, named with lowercase_letters.

When creating a SimState through any constructor, you may pass the keyword arguments `add_options` and `remove_options`, which should be sets of options that modify the initial options set from the default.

```
# Example: enable lazy solves, an option that causes state satisfiability to be checked␣
↪as infrequently as possible.
# This change to the settings will be propagated to all successor states created from␣
↪this state after this line.
>>> s.options.add(angr.options.LAZY_SOLVES)

# Create a new state with lazy solves enabled
>>> s = proj.factory.entry_state(add_options={angr.options.LAZY_SOLVES})

# Create a new state without simplification options enabled
>>> s = proj.factory.entry_state(remove_options=angr.options.simplification)
```

### 3.4.6 State Plugins

With the exception of the set of options just discussed, everything stored in a SimState is actually stored in a *plugin* attached to the state. Almost every property on the state we've discussed so far is a plugin - `memory`, `registers`, `mem`, `regs`, `solver`, etc. This design allows for code modularity as well as the ability to easily *implement new kinds of data storage* for other aspects of an emulated state, or the ability to provide alternate implementations of plugins.

For example, the normal `memory` plugin simulates a flat memory space, but analyses can choose to enable the "abstract memory" plugin, which uses alternate data types for addresses to simulate free-floating memory mappings independent of address, to provide `state.memory`. Conversely, plugins can reduce code complexity: `state.memory` and `state.registers` are actually two different instances of the same plugin, since the registers are emulated with an address space as well.

#### The globals plugin

`state.globals` is an extremely simple plugin: it implements the interface of a standard Python dict, allowing you to store arbitrary data on a state.

#### The history plugin

`state.history` is a very important plugin storing historical data about the path a state has taken during execution. It is actually a linked list of several history nodes, each one representing a single round of execution—you can traverse this list with `state.history.parent.parent` etc.

To make it more convenient to work with this structure, the history also provides several efficient iterators over the history of certain values. In general, these values are stored as `history.recent_NAME` and the iterator over them is just `history.NAME`. For example, `for addr in state.history.bbl_addrs: print hex(addr)` will print out a basic block address trace for the binary, while `state.history.recent_bbl_addrs` is the list of basic blocks executed in the most recent step, `state.history.parent.recent_bbl_addrs` is the list of basic blocks executed in the previous step, etc. If you ever need to quickly obtain a flat list of these values, you can access `.hardcopy`, e.g. `state.history.bbl_addrs.hardcopy`. Keep in mind though, index-based accessing is implemented on the iterators.

Here is a brief listing of some of the values stored in the history:

- `history.descriptions` is a listing of string descriptions of each of the rounds of execution performed on the state.
- `history.bbl_addrs` is a listing of the basic block addresses executed by the state. There may be more than one per round of execution, and not all addresses may correspond to binary code - some may be addresses at which SimProcedures are hooked.
- `history.jumpkinds` is a listing of the disposition of each of the control flow transitions in the state's history, as VEX enum strings.
- `history.jump_guards` is a listing of the conditions guarding each of the branches that the state has encountered.
- `history.events` is a semantic listing of "interesting events" which happened during execution, such as the presence of a symbolic jump condition, the program popping up a message box, or execution terminating with an exit code.
- `history.actions` is usually empty, but if you add the `angr.options.refs` options to the state, it will be populated with a log of all the memory, register, and temporary value accesses performed by the program.

#### The callstack plugin

angr will track the call stack for the emulated program. On every call instruction, a frame will be added to the top of the tracked callstack, and whenever the stack pointer drops below the point where the topmost frame was called, a frame is popped. This allows angr to robustly store data local to the current emulated function.

Similar to the history, the callstack is also a linked list of nodes, but there are no provided iterators over the contents of the nodes - instead you can directly iterate over `state.callstack` to get the callstack frames for each of the active frames, in order from most recent to oldest. If you just want the topmost frame, this is `state.callstack`.

- `callstack.func_addr` is the address of the function currently being executed
- `callstack.call_site_addr` is the address of the basic block which called the current function
- `callstack.stack_ptr` is the value of the stack pointer from the beginning of the current function
- `callstack.ret_addr` is the location that the current function will return to if it returns

### 3.4.7 More about I/O: Files, file systems, and network sockets

Please refer to *Working with File System, Sockets, and Pipes* for a more complete and detailed documentation of how I/O is modeled in angr.

### 3.4.8 Copying and Merging

A state supports very fast copies, so that you can explore different possibilities:

```
>>> proj = angr.Project('/bin/true')
>>> s = proj.factory.blank_state()
>>> s1 = s.copy()
>>> s2 = s.copy()

>>> s1.mem[0x1000].uint32_t = 0x41414141
>>> s2.mem[0x1000].uint32_t = 0x42424242
```

States can also be merged together.

```
# merge will return a tuple. the first element is the merged state
# the second element is a symbolic variable describing a state flag
# the third element is a boolean describing whether any merging was done
>>> (s_merged, m, anything_merged) = s1.merge(s2)

# this is now an expression that can resolve to "AAAA" *or* "BBBB"
>>> aaaa_or_bbbb = s_merged.mem[0x1000].uint32_t
```

**Todo**

describe limitations of merging

## 3.5 Simulation Managers

The most important control interface in angr is the SimulationManager, which allows you to control symbolic execution over groups of states simultaneously, applying search strategies to explore a program's state space. Here, you'll learn how to use it.

Simulation managers let you wrangle multiple states in a slick way. States are organized into "stashes", which you can step forward, filter, merge, and move around as you wish. This allows you to, for example, step two different stashes of states at different rates, then merge them together. The default stash for most operations is the `active` stash, which is where your states get put when you initialize a new simulation manager.

### 3.5.1 Stepping

The most basic capability of a simulation manager is to step forward all states in a given stash by one basic block. You do this with `.step()`.

```
>>> import angr
>>> proj = angr.Project('examples/fauxware/fauxware', auto_load_libs=False)
>>> state = proj.factory.entry_state()
>>> simgr = proj.factory.simgr(state)
>>> simgr.active
[<SimState @ 0x400580>]

>>> simgr.step()
>>> simgr.active
[<SimState @ 0x400540>]
```

Of course, the real power of the stash model is that when a state encounters a symbolic branch condition, both of the successor states appear in the stash, and you can step both of them in sync. When you don't really care about controlling analysis very carefully and you just want to step until there's nothing left to step, you can just use the `.run()` method.

```
# Step until the first symbolic branch
>>> while len(simgr.active) == 1:
...     simgr.step()

>>> simgr
<SimulationManager with 2 active>
>>> simgr.active
[<SimState @ 0x400692>, <SimState @ 0x400699>]

# Step until everything terminates
>>> simgr.run()
>>> simgr
<SimulationManager with 3 deadended>
```

We now have 3 deadended states! When a state fails to produce any successors during execution, for example, because it reached an `exit` syscall, it is removed from the active stash and placed in the `deadended` stash.

### 3.5.2 Stash Management

Let's see how to work with other stashes.

To move states between stashes, use `.move()`, which takes `from_stash`, `to_stash`, and `filter_func` (optional, default is to move everything). For example, let's move everything that has a certain string in its output:

```
>>> simgr.move(from_stash='deadended', to_stash='authenticated', filter_func=lambda s: b
↪'Welcome' in s.posix.dumps(1))
>>> simgr
<SimulationManager with 2 authenticated, 1 deadended>
```

We were able to just create a new stash named "authenticated" just by asking for states to be moved to it. All the states in this stash have "Welcome" in their stdout, which is a fine metric for now.

Each stash is just a list, and you can index into or iterate over the list to access each of the individual states, but there are some alternate methods to access the states too. If you prepend the name of a stash with `one_`, you will be given the first state in the stash. If you prepend the name of a stash with `mp_`, you will be given a mulpyplexed version of the stash.

```
>>> for s in simgr.deadended + simgr.authenticated:
...     print(hex(s.addr))
0x1000030
0x1000078
0x1000078

>>> simgr.one_deadended
<SimState @ 0x1000030>
>>> simgr.mp_authenticated
MP([<SimState @ 0x1000078>, <SimState @ 0x1000078>])
>>> simgr.mp_authenticated.posix.dumps(0)
MP(['\x00\x00\x00\x00\x00\x00\x00\x00\x00SOSNEAKY\x00',
    '\x00\x00\x00\x00\x00\x00\x00\x00\x00S\x80\x80\x80\x80@\x80@\x00'])
```

Of course, `step`, `run`, and any other method that operates on a single stash of paths can take a `stash` argument, specifying which stash to operate on.

There are lots of fun tools that the simulation manager provides you for managing your stashes. We won't go into the rest of them for now, but you should check out the API documentation.

#### Stash types

You can use stashes for whatever you like, but there are a few stashes that will be used to categorize some special kinds of states. These are:

| Stash | Description |
|---|---|
| active | This stash contains the states that will be stepped by default, unless an alternate stash is specified. |
| deadended | A state goes to the deadended stash when it cannot continue the execution for some reason, including no more valid instructions, unsat state of all of its successors, or an invalid instruction pointer. |
| pruned | When using `LAZY_SOLVES`, states are not checked for satisfiability unless absolutely necessary. When a state is found to be unsat in the presence of `LAZY_SOLVES`, the state hierarchy is traversed to identify when, in its history, it initially became unsat. All states that are descendants of that point (which will also be unsat, since a state cannot become un-unsat) are pruned and put in this stash. |
| unconstrained | If the `save_unconstrained` option is provided to the SimulationManager constructor, states that are determined to be unconstrained (i.e., with the instruction pointer controlled by user data or some other source of symbolic data) are placed here. |
| unsat | If the `save_unsat` option is provided to the SimulationManager constructor, states that are determined to be unsatisfiable (i.e., they have constraints that are contradictory, like the input having to be both "AAAA" and "BBBB" at the same time) are placed here. |

There is another list of states that is not a stash: `errored`. If, during execution, an error is raised, then the state will be wrapped in an `ErrorRecord` object, which contains the state and the error it raised, and then the record will be inserted into `errored`. You can get at the state as it was at the beginning of the execution tick that caused the error with `record.state`, you can see the error that was raised with `record.error`, and you can launch a debug shell at the site of the error with `record.debug()`. This is an invaluable debugging tool!

### 3.5.3 Simple Exploration

An extremely common operation in symbolic execution is to find a state that reaches a certain address, while discarding all states that go through another address. Simulation manager has a shortcut for this pattern, the `.explore()` method.

When launching `.explore()` with a `find` argument, execution will run until a state is found that matches the find condition, which can be the address of an instruction to stop at, a list of addresses to stop at, or a function which takes

a state and returns whether it meets some criteria. When any of the states in the active stash match the `find` condition, they are placed in the `found` stash, and execution terminates. You can then explore the found state, or decide to discard it and continue with the other ones. You can also specify an `avoid` condition in the same format as `find`. When a state matches the avoid condition, it is put in the `avoided` stash, and execution continues. Finally, the `num_find` argument controls the number of states that should be found before returning, with a default of 1. Of course, if you run out of states in the active stash before finding this many solutions, execution will stop anyway.

Let's look at a simple crackme example:

First, we load the binary.

```
>>> proj = angr.Project('examples/CSCI-4968-MBE/challenges/crackme0x00a/crackme0x00a')
```

Next, we create a SimulationManager.

```
>>> simgr = proj.factory.simgr()
```

Now, we symbolically execute until we find a state that matches our condition (i.e., the "win" condition).

```
>>> simgr.explore(find=lambda s: b"Congrats" in s.posix.dumps(1))
<SimulationManager with 1 active, 1 found>
```

Now, we can get the flag out of that state!

```
>>> s = simgr.found[0]
>>> print(s.posix.dumps(1))
Enter password: Congrats!

>>> flag = s.posix.dumps(0)
>>> print(flag)
g00dJ0B!
```

Pretty simple, isn't it?

Other examples can be found by browsing the *examples*.

### Exploration Techniques

angr ships with several pieces of canned functionality that let you customize the behavior of a simulation manager, called *exploration techniques*. The archetypical example of why you would want an exploration technique is to modify the pattern in which the state space of the program is explored - the default "step everything at once" strategy is effectively breadth-first search, but with an exploration technique you could implement, for example, depth-first search. However, the instrumentation power of these techniques is much more flexible than that - you can totally alter the behavior of angr's stepping process. Writing your own exploration techniques will be covered in a later chapter.

To use an exploration technique, call `simgr.use_technique(tech)`, where tech is an instance of an ExplorationTechnique subclass. angr's built-in exploration techniques can be found under `angr.exploration_techniques`.

Here's a quick overview of some of the built-in ones:

- *DFS*: Depth first search, as mentioned earlier. Keeps only one state active at once, putting the rest in the `deferred` stash until it deadends or errors.
- *Explorer*: This technique implements the `.explore()` functionality, allowing you to search for and avoid addresses.
- *LengthLimiter*: Puts a cap on the maximum length of the path a state goes through.

- *LoopSeer*: Uses a reasonable approximation of loop counting to discard states that appear to be going through a loop too many times, putting them in a `spinning` stash and pulling them out again if we run out of otherwise viable states.
- *ManualMergepoint*: Marks an address in the program as a merge point, so states that reach that address will be briefly held, and any other states that reach that same point within a timeout will be merged together.
- *MemoryWatcher*: Monitors how much memory is free/available on the system between simgr steps and stops exploration if it gets too low.
- *Oppologist*: The "operation apologist" is an especially fun gadget - if this technique is enabled and angr encounters an unsupported instruction, for example a bizarre and foreign floating point SIMD op, it will concretize all the inputs to that instruction and emulate the single instruction using the unicorn engine, allowing execution to continue.
- *Spiller*: When there are too many states active, this technique can dump some of them to disk in order to keep memory consumption low.
- *Threading*: Adds thread-level parallelism to the stepping process. This doesn't help much because of Python's global interpreter locks, but if you have a program whose analysis spends a lot of time in angr's native-code dependencies (unicorn, z3, libvex) you can seem some gains.
- *Tracer*: An exploration technique that causes execution to follow a dynamic trace recorded from some other source. The dynamic tracer repository has some tools to generate those traces.
- *Veritesting*: An implementation of a CMU paper on automatically identifying useful merge points. This is so useful, you can enable it automatically with `veritesting=True` in the SimulationManager constructor! Note that it frequenly doesn't play nice with other techniques due to the invasive way it implements static symbolic execution.

Look at the API documentation for the *SimulationManager* and `ExplorationTechnique` classes for more information.

## 3.6 Simulation and Instrumentation

When you ask for a step of execution to happen in angr, something has to actually perform the step. angr uses a series of engines (subclasses of the `SimEngine` class) to emulate the effects that of a given section of code has on an input state. The execution core of angr simply tries all the available engines in sequence, taking the first one that is able to handle the step. The following is the default list of engines, in order:

- The failure engine kicks in when the previous step took us to some uncontinuable state
- The syscall engine kicks in when the previous step ended in a syscall
- The hook engine kicks in when the current address is hooked
- The unicorn engine kicks in when the `UNICORN` state option is enabled and there is no symbolic data in the state
- The VEX engine kicks in as the final fallback.

### 3.6.1 SimSuccessors

The code that actually tries all the engines in turn is `project.factory.successors(state, **kwargs)`, which passes its arguments onto each of the engines. This function is at the heart of `state.step()` and `simulation_manager.step()`. It returns a SimSuccessors object, which we discussed briefly before. The purpose of SimSuccessors is to perform a simple categorization of the successor states, stored in various list attributes. They are:

| Attribut | Guard Condition | Instruction Pointer | Description |
|---|---|---|---|
| succ | True (can be symbolic, but constrained to True) | Can be symbolic (but 256 solutions or less; see unconstr | A normal, satisfiable successor state to the state processed by the engine. The instruction pointer of this state may be symbolic (i.e., a computed jump based on user input), so the state might actually represent *several* potential continuations of execution going forward. |
| unsa | False (can be symbolic, but constrained to False). | Can be symbolic. | Unsatisfiable successors. These are successors whose guard conditions can only be false (i.e., jumps that cannot be taken, or the default branch of jumps that *must* be taken). |
| flat | True (can be symbolic, but constrained to True). | Concrete value. | As noted above, states in the `successors` list can have symbolic instruction pointers. This is rather confusing, as elsewhere in the code (i.e., in `SimEngineVEX.process`, when it's time to step that state forward), we make assumptions that a single program state only represents the execution of a single spot in the code. To alleviate this, when we encounter states in `successors` with symbolic instruction pointers, we compute all possible concrete solutions (up to an arbitrary threshold of 256) for them, and make a copy of the state for each such solution. We call this process "flattening". These `flat_successors` are states, each of which has a different, concrete instruction pointer. For example, if the instruction pointer of a state in `successors` was `X+5`, where `X` had constraints of `X > 0x800000` and `X <= 0x800010`, we would flatten it into 16 different `flat_successors` states, one with an instruction pointer of `0x800006`, one with `0x800007`, and so on until `0x800015`. |
| unco | True (can be symbolic, but constrained to True). | Symbolic (with more than 256 solutions). | During the flattening procedure described above, if it turns out that there are more than 256 possible solutions for the instruction pointer, we assume that the instruction pointer has been overwritten with unconstrained data (i.e., a stack overflow with user data). *This assumption is not sound in general*. Such states are placed in `unconstrained_successors` and not in `successors`. |
| all_ | Anything | Can be symbolic. | This is `successors + unsat_successors + unconstrained_successors`. |

### 3.6.2 Breakpoints

**Todo**

rewrite this to fix the narrative

Like any decent execution engine, angr supports breakpoints. This is pretty cool! A point is set as follows:

```
>>> import angr
>>> b = angr.Project('examples/fauxware/fauxware')

# get our state
>>> s = b.factory.entry_state()

# add a breakpoint. This breakpoint will drop into ipdb right before a memory write␣
↪happens.
>>> s.inspect.b('mem_write')

# on the other hand, we can have a breakpoint trigger right *after* a memory write␣
↪happens.
# we can also have a callback function run instead of opening ipdb.
>>> def debug_func(state):
...     print(f"State {state} is about to do a memory write!")

>>> s.inspect.b('mem_write', when=angr.BP_AFTER, action=debug_func)

# or, you can have it drop you in an embedded IPython!
>>> s.inspect.b('mem_write', when=angr.BP_AFTER, action=angr.BP_IPYTHON)
```

There are many other places to break than a memory write. Here is the list. You can break at BP_BEFORE or BP_AFTER for each of these events.

| Event type | Event meaning |
|---|---|
| mem_read | Memory is being read. |
| mem_write | Memory is being written. |
| address_concretization | A symbolic memory access is being resolved. |
| reg_read | A register is being read. |
| reg_write | A register is being written. |
| tmp_read | A temp is being read. |
| tmp_write | A temp is being written. |
| expr | An expression is being created (i.e., a result of an arithmetic operation or a constant in the IR). |
| statement | An IR statement is being translated. |
| instruction | A new (native) instruction is being translated. |
| irsb | A new basic block is being translated. |
| constraints | New constraints are being added to the state. |
| exit | A successor is being generated from execution. |
| fork | A symbolic execution state has forked into multiple states. |
| symbolic_variable | A new symbolic variable is being created. |
| call | A call instruction is hit. |
| return | A ret instruction is hit. |
| simprocedure | A simprocedure (or syscall) is executed. |
| dirty | A dirty IR callback is executed. |
| syscall | A syscall is executed (called in addition to the simprocedure event). |
| engine_process | A SimEngine is about to process some code. |

These events expose different attributes:

| Event type | Attribute name | Attribute availability | Attribute meaning |
|---|---|---|---|
| mem_read | mem_read_address | BP_BEFORE or BP_AFTER | The address at which memory is being read. |
| mem_read | mem_read_expr | BP_AFTER | The expression at that address. |
| mem_read | mem_read_length | BP_BEFORE or BP_AFTER | The length of the memory read. |
| mem_read | mem_read_condition | BP_BEFORE or BP_AFTER | The condition of the memory read. |
| mem_write | mem_write_address | BP_BEFORE or BP_AFTER | The address at which memory is being written. |
| mem_write | mem_write_length | BP_BEFORE or BP_AFTER | The length of the memory write. |
| mem_write | mem_write_expr | BP_BEFORE or BP_AFTER | The expression that is being written. |
| mem_write | mem_write_condition | BP_BEFORE or BP_AFTER | The condition of the memory write. |
| reg_read | reg_read_offset | BP_BEFORE or BP_AFTER | The offset of the register being read. |
| reg_read | reg_read_length | BP_BEFORE or BP_AFTER | The length of the register read. |
| reg_read | reg_read_expr | BP_AFTER | The expression in the register. |

continues on next page

Table 1 – continued from previous page

| Event type | Attribute name | Attribute availability | Attribute meaning |
|---|---|---|---|
| reg_read | reg_read_condition | BP_BEFORE or BP_AFTER | The condition of the register read. |
| reg_write | reg_write_offset | BP_BEFORE or BP_AFTER | The offset of the register being written. |
| reg_write | reg_write_length | BP_BEFORE or BP_AFTER | The length of the register write. |
| reg_write | reg_write_expr | BP_BEFORE or BP_AFTER | The expression that is being written. |
| reg_write | reg_write_condition | BP_BEFORE or BP_AFTER | The condition of the register write. |
| tmp_read | tmp_read_num | BP_BEFORE or BP_AFTER | The number of the temp being read. |
| tmp_read | tmp_read_expr | BP_AFTER | The expression of the temp. |
| tmp_write | tmp_write_num | BP_BEFORE or BP_AFTER | The number of the temp written. |
| tmp_write | tmp_write_expr | BP_AFTER | The expression written to the temp. |
| expr | expr | BP_BEFORE or BP_AFTER | The IR expression. |
| expr | expr_result | BP_AFTER | The value (e.g. AST) which the expression was evaluated to. |
| statement | statement | BP_BEFORE or BP_AFTER | The index of the IR statement (in the IR basic block). |
| instruction | instruction | BP_BEFORE or BP_AFTER | The address of the native instruction. |
| irsb | address | BP_BEFORE or BP_AFTER | The address of the basic block. |
| constraints | added_constraints | BP_BEFORE or BP_AFTER | The list of constraint expressions being added. |
| call | function_address | BP_BEFORE or BP_AFTER | The name of the function being called. |
| exit | exit_target | BP_BEFORE or BP_AFTER | The expression representing the target of a SimExit. |
| exit | exit_guard | BP_BEFORE or BP_AFTER | The expression representing the guard of a SimExit. |
| exit | exit_jumpkind | BP_BEFORE or BP_AFTER | The expression representing the kind of SimExit. |
| symbolic_variable | symbolic_name | BP_AFTER | The name of the symbolic variable being created. The solver engine might modify this name (by appending a unique ID and length). Check the symbolic_expr for the final symbolic expression. |
| symbolic_variable | symbolic_size | BP_AFTER | The size of the symbolic variable being created. |

continues on next page

Table 1 – continued from previous page

| Event type | Attribute name | Attribute availability | Attribute meaning |
|---|---|---|---|
| symbolic_variable | symbolic_expr | BP_AFTER | The expression representing the new symbolic variable. |
| address_concretization | address_concretization_strateg | BP_BEFORE or BP_AFTER | The SimConcretizationStrategy being used to resolve the address. This can be modified by the breakpoint handler to change the strategy that will be applied. If your breakpoint handler sets this to None, this strategy will be skipped. |
| address_concretization | address_concretization_action | BP_BEFORE or BP_AFTER | The SimAction object being used to record the memory action. |
| address_concretization | address_concretization_memo | BP_BEFORE or BP_AFTER | The SimMemory object on which the action was taken. |
| address_concretization | address_concretization_expr | BP_BEFORE or BP_AFTER | The AST representing the memory index being resolved. The breakpoint handler can modify this to affect the address being resolved. |
| address_concretization | address_concretization_add_c | BP_BEFORE or BP_AFTER | Whether or not constraints should/will be added for this read. |
| address_concretization | address_concretization_result | BP_AFTER | The list of resolved memory addresses (integers). The breakpoint handler can overwrite these to effect a different resolution result. |
| syscall | syscall_name | BP_BEFORE or BP_AFTER | The name of the system call. |
| simprocedure | simprocedure_name | BP_BEFORE or BP_AFTER | The name of the simprocedure. |
| simprocedure | simprocedure_addr | BP_BEFORE or BP_AFTER | The address of the simprocedure. |
| simprocedure | simprocedure_result | BP_AFTER | The return value of the simprocedure. You can also *override* it in BP_BEFORE, which will cause the actual simprocedure to be skipped and for your return value to be used instead. |
| simprocedure | simprocedure | BP_BEFORE or BP_AFTER | The actual SimProcedure object. |

continues on next page

Table 1 – continued from previous page

| Event type | Attribute name | Attribute availability | Attribute meaning |
|---|---|---|---|
| dirty | dirty_name | BP_BEFORE or BP_AFTER | The name of the dirty call. |
| dirty | dirty_handler | BP_BEFORE | The function that will be run to handle the dirty call. You can override this. |
| dirty | dirty_args | BP_BEFORE or BP_AFTER | The address of the dirty. |
| dirty | dirty_result | BP_AFTER | The return value of the dirty call. You can also *override* it in BP_BEFORE, which will cause the actual dirty call to be skipped and for your return value to be used instead. |
| engine_process | sim_engine | BP_BEFORE or BP_AFTER | The SimEngine that is processing. |
| engine_process | successors | BP_BEFORE or BP_AFTER | The SimSuccessors object defining the result of the engine. |

These attributes can be accessed as members of `state.inspect.attrs` during the appropriate breakpoint callback to access the appropriate values. You can even modify these value to modify further uses of the values!

```
>>> def track_reads(state):
...     print('Read', state.inspect.attrs.mem_read_expr, 'from', state.inspect.attrs.mem_
↪read_address)
...
>>> s.inspect.b('mem_read', when=angr.BP_AFTER, action=track_reads)
```

Additionally, each of these properties can be used as a keyword argument to `inspect.b` to make the breakpoint conditional:

```
# This will break before a memory write if 0x1000 is a possible value of its target␣
↪expression
>>> s.inspect.b('mem_write', mem_write_address=0x1000)

# This will break before a memory write if 0x1000 is the *only* value of its target␣
↪expression
>>> s.inspect.b('mem_write', mem_write_address=0x1000, mem_write_address_unique=True)

# This will break after instruction 0x8000, but only 0x1000 is a possible value of the␣
↪last expression that was read from memory
>>> s.inspect.b('instruction', when=angr.BP_AFTER, instruction=0x8000, mem_read_
↪expr=0x1000)
```

Cool stuff! In fact, we can even specify a function as a condition:

```
# this is a complex condition that could do anything! In this case, it makes sure that␣
↪RAX is 0x41414141 and
# that the basic block starting at 0x8004 was executed sometime in this path's history
```

(continues on next page)

(continued from previous page)

```
>>> def cond(state):
...     return state.eval(state.regs.rax, cast_to=str) == 'AAAA' and 0x8004 in state.
↪inspect.attrs.backtrace

>>> s.inspect.b('mem_write', condition=cond)
```

That is some cool stuff!

#### Caution about `mem_read` breakpoint

The `mem_read` breakpoint gets triggered anytime there are memory reads by either the executing program or the binary analysis. If you are using breakpoint on `mem_read` and also using `state.mem` to load data from memory addresses, then know that the breakpoint will be fired as you are technically reading memory.

So if you want to load data from memory and not trigger any `mem_read` breakpoint you have had set up, then use `state.memory.load` with the keyword arguments `disable_actions=True` and `inspect=False`.

This is also true for `state.find` and you can use the same keyword arguments to prevent `mem_read` breakpoints from firing.

## 3.7 Analyses

angr's goal is to make it easy to carry out useful analyses on binary programs. To this end, angr allows you to package analysis code in a common format that can be easily applied to any project. We will cover writing your own analyses *Writing Analyses*, but the idea is that all the analyses appear under `project.analyses` (for example, `project.analyses.CFGFast()`) and can be called as functions, returning analysis result instances.

### 3.7.1 Built-in Analyses

| Name | Description |
|---|---|
| CFGFast | Constructs a fast *Control Flow Graph* of the program |
| CFGEmulated | Constructs an accurate *Control Flow Graph* of the program |
| VFG | Performs VSA on every function of the program, creating a *Value Flow Graph* and detecting stack variables |
| DDG | Calculates a *Data Dependency Graph*, allowing one to determine what statements a given value depends on |
| BackwardSlice | Computes a *Backward Slice* of a program with respect to a certain target |
| Identifier | Identifies common library functions in CGC binaries |
| More! | angr has quite a few analyses, most of which work! If you'd like to know how to use one, please submit an issue requesting documentation. |

### 3.7.2 Resilience

Analyses can be written to be resilient, and catch and log basically any error. These errors, depending on how they're caught, are logged to the `errors` or `named_errors` attribute of the analysis. However, you might want to run an analysis in "fail fast" mode, so that errors are not handled. To do this, the argument `fail_fast=True` can be passed into the analysis constructor.

## 3.8 Symbolic Execution

Symbolic execution is a program analysis technique used to explore multiple execution paths of a program simultaneously. Unlike normal execution, which runs the program with specific inputs, symbolic execution treats inputs as symbolic variables rather than concrete values. This means the execution can represent a wide range of inputs with symbolic expressions. Symbolic execution allows, at a time in emulation, to determine for a branch all conditions necessary to take a branch or not. Every variable is represented as a symbolic value, and each branch as a constraint. Thus, symbolic execution allows us to see which conditions allow the program to go from point A to point B by resolving these constraints. The execution paths are then analyzed by solving constraints generated by these symbolic expressions, allowing the discovery of bugs and vulnerabilities that might be missed in standard testing.

### 3.8.1 Example:

Consider the following simple program :

```
const char* check_value(int x) {
    if (x > 10) {
        return "Greater";
    } else {
        return "Lesser or Equal";
    }
}
```

In normal execution, if `x` is set to 5, the program will follow the path where `x <= 10` and return "Lesser or Equal". In symbolic execution, `x` is treated as a symbolic variable, `X`. The execution engine explores both paths:

- Path 1: `X > 10` leading to the result "Greater"
- Path 2: `X <= 10` leading to the result "Lesser or Equal"

Constraints for both paths are generated and solved to understand all possible behaviors of the program.

In software verification, it helps ensure that the code behaves as expected across all possible inputs and states. For security analysis, symbolic execution can uncover vulnerabilities such as input validation errors, which could be exploited by attackers. Additionally, in automated testing, it aids in generating comprehensive test cases that cover edge cases and rare execution paths, enhancing the robustness and security of software systems. Overall, symbolic execution provides a powerful means to rigorously analyze and improve software and firmware reliability.

### 3.8.2 Basic Execution

Now let's see an example use case of symbolic execution with angr. Consider the following example code,

```
void helloWorld() {
    printf("Hello, World!\n");
}


void firstCall(uint32_t num) {
    if (num > 50 && num <100)
        HelloWorld();
}
```

The `firstCall` function will accept a 32-bit number as an input and will call `helloWorld` function if the number is between 50 and 100.

You can perform a symbolic execution to find a correct and valid input to reach the final `helloWorld` function call with angr using the following sample code.

```
import angr, claripy
# Load the binary
project = angr.Project('./3func', auto_load_libs=False)

# Define the address of the firstCall function
firstCall_addr = project.loader.main_object.get_symbol("firstCall")

# Define the address of the helloWorld function
helloWorld_addr = project.loader.main_object.get_symbol("helloWorld")
# Create a symbolic variable for the firstCall arg
input_arg = claripy.BVS('input_arg', 32)

# Create a blank state at the address of the firstCall function
init_state = project.factory.blank_state(addr=firstCall_addr.rebased_addr)

# Assuming the calling convention passes the argument in a register
# (e.g., x86 uses edi for the argument)
init_state.regs.edi = input_arg

# Create a simulation manager
simgr = project.factory.simulation_manager(init_state)

# Explore the binary, looking for the address of helloWorld
simgr.explore(find=helloWorld_addr.rebased_addr)

# Check if we found a state that reached the target
if simgr.found:
    input_value = simgr.found[0].solver.eval(input_arg)
    print(f"Value of input_arg that reaches HelloWorld: {input_value}")
    # Get the constraints for reaching the helloWorld function
    constraints = simgr.found[0].solver.constraints
    # Create a solver with the constraints
    solver = claripy.Solver()
    solver.add(constraints)
    min_val = solver.min(input_arg)
    max_val = solver.max(input_arg)
    print(f"Function arg: min = {min_val}, max = {max_val}")
else:
    print("Did not find a state that reaches HelloWorld.")
```

It will produce the output like the below with a valid example function arg that can reach the function `helloWorld` that you can use as a test case.

```
Value of input_arg that reaches HelloWorld: 71
Function arg: min = 51, max = 99
```

## 3.9 A final word of advice

Congratulations! If you've read this far through the book (editor's note: this comment only really applies when we've actually finished writing all the TODOs so far) then you've been introduced to all the fundamental components of angr necessary to get started with binary analysis.

Ultimately, angr is just an emulator. It is a highly instrumentable and very unique emulator with lots of considerations

for environment, true, but at its core, the work you do with angr is about extracting knowledge about how a bunch of bytecode behaves on a CPU. In designing angr, we've tried to provide you with the tools and abstractions on top of this emulator to make certain common tasks more useful, but there's no problem you can't solve just by working with a SimState and observing the affects of `.step()`.

As you read further into this book, we'll describe more technical subjects and how to tune angr's behavior for complicated scenarios. This knowledge should inform your use of angr so you can take the quickest path to a solution to any given problem, but ultimately, you will want to solve problems by exercising creativity with the tools at your disposal. If you can take a problem and wrangle it into a form where it has defined and tractable inputs and outputs, you can absolutely use angr to achieve your goals, given that these goals involve analyzing binaries. None of the abstractions or instrumentations we provide are the end-all of how to use angr for a given task - angr is designed so it can be used in as integrated or as ad-hoc of a manner as you desire. If you see a path from problem to solution, take it.

Of course, it's very difficult to become well-acquainted with such a huge piece of technology as angr. To this end you can absolutely lean on the community (through the angr Discord server is the best option) to discuss angr and solving problems with it.

Good luck!

CHAPTER

FOUR

# BUILD-IN ANALYSES

## 4.1 Control-flow Graph Recovery (CFG)

angr includes analyses to recover the control-flow graph of a binary program. This also includes recovery of function boundaries, as well as reasoning about indirect jumps and other useful metadata.

### 4.1.1 General ideas

A basic analysis that one might carry out on a binary is a Control Flow Graph. A CFG is a graph with (conceptually) basic blocks as nodes and jumps/calls/rets/etc as edges.

In angr, there are two types of CFG that can be generated: a static CFG (CFGFast) and a dynamic CFG (CFGEmulated).

CFGFast uses static analysis to generate a CFG. It is significantly faster, but is theoretically bounded by the fact that some control-flow transitions can only be resolved at execution-time. This is the same sort of CFG analysis performed by other popular reverse-engineering tools, and its results are comparable with their output.

CFGEmulated uses symbolic execution to capture the CFG. While it is theoretically more accurate, it is dramatically slower. It is also typically less complete, due to issues with the accuracy of emulation (system calls, missing hardware features, and so on)

*If you are unsure which CFG to use, or are having problems with CFGEmulated, try CFGFast first.*

A CFG can be constructed by doing:

```
>>> import angr
# load your project
>>> p = angr.Project('/bin/true', load_options={'auto_load_libs': False})

# Generate a static CFG
>>> cfg = p.analyses.CFGFast()

# generate a dynamic CFG
>>> cfg = p.analyses.CFGEmulated(keep_state=True)
```

### 4.1.2 Using the CFG

The CFG, at its core, is a NetworkX di-graph. This means that all of the normal NetworkX APIs are available:

```
>>> print("This is the graph:", cfg.model.graph)
>>> print("It has %d nodes and %d edges" % (len(cfg.model.graph.nodes()), len(cfg.model.
↪graph.edges())))
```

The nodes of the CFG graph are instances of class `CFGNode`. Due to context sensitivity, a given basic block can have multiple nodes in the graph (for multiple contexts).

```
# this grabs *any* node at a given location:
>>> entry_node = cfg.model.get_any_node(p.entry)

# on the other hand, this grabs all of the nodes
>>> print("There were %d contexts for the entry block" % len(cfg.model.get_all_nodes(p.
↪entry)))

# we can also look up predecessors and successors
>>> print("Predecessors of the entry point:", entry_node.predecessors)
>>> print("Successors of the entry point:", entry_node.successors)
>>> print("Successors (and type of jump) of the entry point:", [ jumpkind + " to " +␣
↪str(node.addr) for node,jumpkind in cfg.model.get_successors_and_jumpkind(entry_node)␣
↪])
```

#### Viewing the CFG

Control-flow graph rendering is a hard problem. angr does not provide any built-in mechanism for rendering the output of a CFG analysis, and attempting to use a traditional graph rendering library, like matplotlib, will result in an unusable image.

One solution for viewing angr CFGs is found in axt's angr-utils repository.

### 4.1.3 Shared Libraries

The CFG analysis does not distinguish between code from different binary objects. This means that by default, it will try to analyze control flow through loaded shared libraries. This is almost never intended behavior, since this will extend the analysis time to several days, probably. To load a binary without shared libraries, add the following keyword argument to the `Project` constructor: `load_options={'auto_load_libs': False}`

### 4.1.4 Function Manager

The CFG result produces an object called the *Function Manager*, accessible through `cfg.kb.functions`. The most common use case for this object is to access it like a dictionary. It maps addresses to `Function` objects, which can tell you properties about a function.

```
>>> entry_func = cfg.kb.functions[p.entry]
```

Functions have several important properties!

- `entry_func.block_addrs` is a set of addresses at which basic blocks belonging to the function begin.
- `entry_func.blocks` is the set of basic blocks belonging to the function, that you can explore and disassemble using capstone.
- `entry_func.string_references()` returns a list of all the constant strings that were referred to at any point in the function. They are formatted as `(addr, string)` tuples, where addr is the address in the binary's data section the string lives, and string is a Python string that contains the value of the string.
- `entry_func.returning` is a boolean value signifying whether or not the function can return. `False` indicates that all paths do not return.
- `entry_func.callable` is an angr Callable object referring to this function. You can call it like a Python function with Python arguments and get back an actual result (may be symbolic) as if you ran the function with those arguments!

- `entry_func.transition_graph` is a NetworkX DiGraph describing control flow within the function itself. It resembles the control-flow graphs IDA displays on a per-function level.
- `entry_func.name` is the name of the function.
- `entry_func.has_unresolved_calls` and `entry.has_unresolved_jumps` have to do with detecting imprecision within the CFG. Sometimes, the analysis cannot detect what the possible target of an indirect call or jump could be. If this occurs within a function, that function will have the appropriate `has_unresolved_*` value set to `True`.
- `entry_func.get_call_sites()` returns a list of all the addresses of basic blocks which end in calls out to other functions.
- `entry_func.get_call_target(callsite_addr)` will, given `callsite_addr` from the list of call site addresses, return where that callsite will call out to.
- `entry_func.get_call_return(callsite_addr)` will, given `callsite_addr` from the list of call site addresses, return where that callsite should return to.

and many more !

### 4.1.5 CFGFast details

CFGFast performs a static control-flow and function recovery. Starting with the entry point (or any user-defined points) roughly the following procedure is performed:

1) The basic block is lifted to VEX IR, and all its exits (jumps, calls, returns, or continuation to the next block) are collected
2) For each exit, if this exit is a constant address, we add an edge to the CFG of the correct type, and add the destination block to the set of blocks to be analyzed.
3) In the event of a function call, the destination block is also considered the start of a new function. If the target function is known to return, the block after the call is also analyzed.
4) In the event of a return, the current function is marked as returning, and the appropriate edges in the callgraph and CFG are updated.
5) For all indirect jumps (block exits with a non-constant destination) Indirect Jump Resolution is performed.

#### Finding function starts

CFGFast supports multiple ways of deciding where a function starts and ends.

First the binary's main entry point will be analyzed. For binaries with symbols (e.g., non-stripped ELF and PE binaries) all function symbols will be used as possible starting points. For binaries without symbols, such as stripped binaries, or binaries loaded using the `blob` loader backend, CFG will scan the binary for a set of function prologues defined for the binary's architecture. Finally, by default, the binary's entire code section will be scanned for executable contents, regardless of prologues or symbols.

In addition to these, as with CFGEmulated, function starts will also be considered when they are the target of a "call" instruction on the given architecture.

All of these options can be disabled

#### FakeRets and function returns

When a function call is observed, we first assume that the callee function eventually returns, and treat the block after it as part of the caller function. This inferred control-flow edge is known as a "FakeRet". If, in analyzing the callee, we find this not to be true, we update the CFG, removing this "FakeRet", and updating the callgraph and function blocks accordingly. As such, the CFG is recovered *twice*. In doing this, the set of blocks in each function, and whether the function returns, can be recovered and propagated directly.

#### Indirect Jump Resolution

**Todo**

Document this.

#### CFGFast Options

These are the most useful options when working with CFGFast:

| Option | Description |
|---|---|
| force_complete_ | (Default: True) Treat the entire binary as code for the purposes of function detection. If you have a blob (e.g., mixed code and data) *you want to turn this off*. |
| function_starts | A list of addresses, to use as entry points into the analysis. |
| normalize | (Default: False) Normalize the resulting functions (e.g., each basic block belongs to at most one function, back-edges point to the start of basic blocks) |
| resolve_indirect_j | (Default: True) Perform additional analysis to attempt to find targets for every indirect jump found during CFG creation. |
| more! | Examine the docstring on p.analyses.CFGFast for more up-to-date options |

### 4.1.6 CFGEmulated details

#### CFGEmulated Options

The most common options for CFGEmulated include:

| Option | Description |
|---|---|
| context_sensitivity_level | This sets the context sensitivity level of the analysis. See the context sensitivity level section below for more information. This is 1 by default. |
| starts | A list of addresses, to use as entry points into the analysis. |
| avoid_runs | A list of addresses to ignore in the analysis. |
| call_depth | Limit the depth of the analysis to some number calls. This is useful for checking which functions a specific function can directly jump to (by setting `call_depth` to 1). |
| initial_state | An initial state can be provided to the CFG, which it will use throughout its analysis. |
| keep_state | To save memory, the state at each basic block is discarded by default. If `keep_state` is True, the state is saved in the CFGNode. |
| enable_symbolic_back_ | Whether to enable an intensive technique for resolving indirect jumps |
| enable_advanced_backv | Whether to enable another intensive technique for resolving direct jumps |
| more! | Examine the docstring on p.analyses.CFGEmulated for more up-to-date options |

#### Context Sensitivity Level

angr constructs a CFG by executing every basic block and seeing where it goes. This introduces some challenges: a basic block can act differently in different *contexts*. For example, if a block ends in a function return, the target of that return will be different, depending on different callers of the function containing that basic block.

The context sensitivity level is, conceptually, the number of such callers to keep on the callstack. To explain this concept, let's look at the following code:

```
void error(char *error)
{
    puts(error);
}

void alpha()
{
    puts("alpha");
    error("alpha!");
}

void beta()
{
    puts("beta");
    error("beta!");
}

void main()
{
    alpha();
    beta();
}
```

The above sample has four call chains: `main>alpha>puts`, `main>alpha>error>puts` and `main>beta>puts`, and `main>beta>error>puts`. While, in this case, angr can probably execute both call chains, this becomes unfeasible for larger binaries. Thus, angr executes the blocks with states limited by the context sensitivity level. That is, each function is re-analyzed for each unique context that it is called in.

For example, the `puts()` function above will be analyzed with the following contexts, given different context sensitivity levels:

| Level | Meaning | Contexts |
|---|---|---|
| 0 | Callee-only | `puts` |
| 1 | One caller, plus callee | `alpha>puts beta>puts error>puts` |
| 2 | Two callers, plus callee | `alpha>error>puts main>alpha>puts beta>error>puts main>beta>puts` |
| 3 | Three callers, plus callee | `main>alpha>error>puts main>alpha>puts main>beta>error>puts main>beta>puts` |

The upside of increasing the context sensitivity level is that more information can be gleaned from the CFG. For example, with context sensitivity of 1, the CFG will show that, when called from `alpha`, `puts` returns to `alpha`, when called from `error`, `puts` returns to `error`, and so forth. With context sensitivity of 0, the CFG simply shows that `puts` returns to `alpha`, `beta`, and `error`. This, specifically, is the context sensitivity level used in IDA. The downside of increasing the context sensitivity level is that it exponentially increases the analysis time.

## 4.2 Backward Slicing

A *program slice* is a subset of statements that is obtained from the original program, usually by removing zero or more statements. Slicing is often helpful in debugging and program understanding. For instance, it's usually easier to locate the source of a variable on a program slice.

A backward slice is constructed from a *target* in the program, and all data flows in this slice end at the *target*.

angr has a built-in analysis, called `BackwardSlice`, to construct a backward program slice. This section will act as a how-to for angr's `BackwardSlice` analysis, and followed by some in-depth discussion over the implementation choices and limitations.

### 4.2.1 First Step First

To build a `BackwardSlice`, you will need the following information as input.

- **Required** CFG. A control flow graph (CFG) of the program. This CFG must be an accurate CFG (CFGEmulated).
- **Required** Target, which is the final destination that your backward slice terminates at.
- **Optional** CDG. A control dependence graph (CDG) derived from the CFG. angr has a built-in analysis `CDG` for that purpose.
- **Optional** DDG. A data dependence graph (DDG) built on top of the CFG. angr has a built-in analysis `DDG` for that purpose.

A `BackwardSlice` can be constructed with the following code:

```
>>> import angr
# Load the project
>>> b = angr.Project("examples/fauxware/fauxware", load_options={"auto_load_libs": False}
↪)

# Generate a CFG first. In order to generate data dependence graph afterwards, you'll␣
↪have to:
# - keep all input states by specifying keep_state=True.
# - store memory, register and temporary values accesses by adding the angr.options.refs␣
↪option set.
# Feel free to provide more parameters (for example, context_sensitivity_level) for CFG
# recovery based on your needs.
>>> cfg = b.analyses.CFGEmulated(keep_state=True,
...                              state_add_options=angr.sim_options.refs,
...                              context_sensitivity_level=2)

# Generate the control dependence graph
>>> cdg = b.analyses.CDG(cfg)

# Build the data dependence graph. It might take a while. Be patient!
>>> ddg = b.analyses.DDG(cfg)

# See where we wanna go... let's go to the exit() call, which is modeled as a
# SimProcedure.
>>> target_func = cfg.kb.functions.function(name="exit")
# We need the CFGNode instance
>>> target_node = cfg.model.get_any_node(target_func.addr)

# Let's get a BackwardSlice out of them!
# ``targets`` is a list of objects, where each one is either a CodeLocation
# object, or a tuple of CFGNode instance and a statement ID. Setting statement
# ID to -1 means the very beginning of that CFGNode. A SimProcedure does not
# have any statement, so you should always specify -1 for it.
>>> bs = b.analyses.BackwardSlice(cfg, cdg=cdg, ddg=ddg, targets=[ (target_node, -1) ])
```

(continues on next page)

(continued from previous page)

```
# Here is our awesome program slice!
>>> print(bs)
```

Sometimes it's difficult to get a data dependence graph, or you may simply want build a program slice on top of a CFG. That's basically why DDG is an optional parameter. You can build a `BackwardSlice` solely based on CFG by doing:

```
>>> bs = b.analyses.BackwardSlice(cfg, control_flow_slice=True)
BackwardSlice (to [(<CFGNode exit (0x10000a0) [0]>, -1)])
```

### 4.2.2 Using The `BackwardSlice` Object

Before you go ahead and use `BackwardSlice` object, you should notice that the design of this class is fairly arbitrary right now, and it is still subject to change in the near future. We'll try our best to keep this documentation up-to-date.

#### Members

After construction, a `BackwardSlice` has the following members which describe a program slice:

| Member | Mode | Meaning |
|---|---|---|
| runs_in_slice | CFG-only | A `networkx.DiGraph` instance showing addresses of blocks and SimProcedures in the program slice, as well as transitions between them |
| cfg_nodes_in_ | CFG-only | A `networkx.DiGraph` instance showing CFGNodes in the program slice and transitions in between |
| chosen_statement | With DDG | A dict mapping basic block addresses to lists of statement IDs that are part of the program slice |
| chosen_exits | With DDG | A dict mapping basic block addresses to a list of "exits". Each exit in the list is a valid transition in the program slice |

Each "exit" in `chosen_exit` is a tuple including a statement ID and a list of target addresses. For example, an "exit" might look like the following:

```
(35, [ 0x400020 ])
```

If the "exit" is the default exit of a basic block, it'll look like the following:

```
("default", [ 0x400085 ])
```

#### Export an Annotated Control Flow Graph

**Todo**

Document this.

#### User-friendly Representation

Take a look at `BackwardSlice.dbg_repr()`!

### 4.2.3 Implementation Choices

**Todo**

Document this.

### 4.2.4 Limitations

#### Completeness

**Todo**

Document this.

#### Soundness

**Todo**

Document this.

## 4.3 Identifier

The identifier uses test cases to identify common library functions in CGC binaries. It prefilters by finding some basic information about stack variables/arguments. The information of about stack variables can be generally useful in other projects.

```
>>> import angr

# get all the matches
>>> p = angr.Project("../binaries/tests/i386/identifiable")
# note analysis is executed via the Identifier call
>>> idfer = p.analyses.Identifier()
>>> for funcInfo in idfer.func_info:
...     print(hex(funcInfo.addr), funcInfo.name)

0x8048e60 memcmp
0x8048ef0 memcpy
0x8048f60 memmove
0x8049030 memset
0x8049320 fdprintf
0x8049a70 sprintf
0x8049f40 strcasecmp
0x804a0f0 strcmp
0x804a190 strcpy
0x804a260 strlen
0x804a3d0 strncmp
0x804a620 strtol
0x804aa00 strtol
0x80485b0 free
0x804aab0 free
0x804aad0 free
```

(continues on next page)

(continued from previous page)

```
0x8048660 malloc
0x80485b0 free
```

## 4.4 angr Decompiler

### 4.4.1 Analysis Passes

| Name | Description | Sub-analysis |
|---|---|---|
| CFG recovery | Recover the control flow graph. | Indirect branch resolving |
| Indirect branch resolving | Resolve the targets of indirect branches. | Jump table resolving |
| Removing alignment blocks | | |
| Calling convention recovery | | |
| Stack pointer analysis | Determine values of stack pointer at each instruction. | |
| IR Lifting | Lift the original representation to AIL, block by block. | |
| AIL graph building | | |
| Rewriting single-target indirect branches | Replace single-target indirect branches with direct branches. | |
| Making return statements | Convert Ijk_Ret jump kinds into AIL Return statements. | |
| Simplifying AIL blocks | Simplify each AIL block. | Constant folding, copy propagation, dead assignment elimination, peephole optimizations |
| Reaching definition analysis | | |
| Constant folding | | |
| Copy propagation | | |
| Dead assignment elimination | | |
| Peephole optimizations | | |
| Simplifying AIL function | Simplify the entire AIL function. | Assignment expression folding, unifying local variables, call expression folding, reaching definition analysis |
| Assignment expression folding | Eliminate variables that are assigned to once and used once. | Copy propagation |
| Unifying local variables | Find local variables that are always equivalent and eliminate redundant copies. | Copy propagation |
| Call expression folding | Fold call expressions into the variable where its return value is stored. | Copy propagation |
| Call site building | Apply calling conventions to each call site and rewrite call statements to ones with arguments | Reaching definition analysis |
| Variable recovery | Identify local and global variables. | |
| Variable type inference | Collect type constraints and infer variable types. | |
| Simplification passes | | |
| Region identification | Identify single-entry, single-exit regions. | |
| Structure analysis | Structure each identified region to create high-level control flow structures. | |
| Code generation | | |

CHAPTER

FIVE

# ADVANCED TOPICS

## 5.1 Gotchas when using angr

This section contains a list of gotchas that users/victims of angr frequently run into.

### 5.1.1 SimProcedure inaccuracy

To make symbolic execution more tractable, angr replaces common library functions with summaries written in Python. We call these summaries SimProcedures. SimProcedures allow us to mitigate path explosion that would otherwise be introduced by, for example, `strlen` running on a symbolic string.

Unfortunately, our SimProcedures are far from perfect. If angr is displaying unexpected behavior, it might be caused by a buggy/incomplete SimProcedure. There are several things that you can do:

1. Disable the SimProcedure (you can exclude specific SimProcedures by passing options to the `angr.Project` class. This has the drawback of likely leading to a path explosion, unless you are very careful about constraining the input to the function in question. The path explosion can be partially mitigated with other angr capabilities (such as Veritesting).
2. Replace the SimProcedure with something written directly to the situation in question. For example, our `scanf` implementation is not complete, but if you just need to support a single, known format string, you can write a hook to do exactly that.
3. Fix the SimProcedure.

### 5.1.2 Unsupported syscalls

System calls are also implemented as SimProcedures. Unfortunately, there are system calls that we have not yet implemented in angr. There are several workarounds for an unsupported system call:

1. Implement the system call.

   **Todo**

   document this process

2. Hook the callsite of the system call (using `project.hook`) to make the required modifications to the state in an ad-hoc way.
3. Use the `state.posix.queued_syscall_returns` list to queue syscall return values. If a return value is queued, the system call will not be executed, and the value will be used instead. Furthermore, a function can be queued instead as the "return value", which will result in that function being applied to the state when the system call is triggered.

### 5.1.3 Symbolic memory model

The default memory model used by angr is inspired by Mayhem. This memory model supports limited symbolic reads and writes. If the memory index of a read is symbolic and the range of possible values of this index is too wide, the index is concretized to a single value. If the memory index of a write is symbolic at all, the index is concretized to a single value. This is configurable by changing the memory concretization strategies of `state.memory`.

### 5.1.4 Symbolic lengths

SimProcedures, and especially system calls such as `read()` and `write()` might run into a situation where the *length* of a buffer is symbolic. In general, this is handled very poorly: in many cases, this length will end up being concretized outright or retroactively concretized in later steps of execution. Even in cases when it is not, the source or destination file might end up looking a bit "weird".

### 5.1.5 Division by Zero

Z3 has some issues with divisions by zero. For example:

```
>>> z = z3.Solver()
>>> a = z3.BitVec('a', 32)
>>> b = z3.BitVec('b', 32)
>>> c = z3.BitVec('c', 32)
>>> z.add(a/b == c)
>>> z.add(b == 0)
>>> z.check()
>>> print(z.model().eval(b), z.model().eval(a/b))
0 4294967295
```

This makes it very difficult to handle certain situations in Claripy. We post-process the VEX IR itself to explicitly check for zero-divisions and create IRSB side-exits corresponding to the exceptional case, but SimProcedures and custom analysis code may let occurrences of zero divisions split through, which will then cause weird issues in your analysis. Be safe — when dividing, add a constraint against the denominator being zero.

## 5.2 Understanding the Execution Pipeline

If you've made it this far you know that at its core, angr is a highly flexible and intensely instrumentable emulator. In order to get the most mileage out of it, you'll want to know what happens at every step of the way when you say `simgr.run()`.

This is intended to be a more advanced document; you'll need to understand the function and intent of `SimulationManager`, `ExplorationTechnique`, `SimState`, and `SimEngine` in order to understand what we're talking about at times! You may want to have the angr source open to follow along with this.

At every step along the way, each function will take `**kwargs` and pass them along to the next function in the hierarchy, so you can pass parameters to any point in the hierarchy and they will trickle down to everything below.

### 5.2.1 Simulation Managers

So you've set your analysis in motion. Time to begin our journey.

#### `run()`

`SimulationManager.run()` takes several optional parameters, all of which control when to break out of the stepping loop. Notably, `n`, and `until`. `n` is used immediately - the run function loops, calling the `step()` function and passing on all its parameters until either `n` steps have happened or some other termination condition has occurred. If `n` is not provided, it defaults to 1, unless an `until` function is provided, in which case there will be no numerical cap on the

loop. Additionally, the stash that is being used is taken into consideration, as if it becomes empty execution must terminate.

So, in summary, when you call `run()`, `step()` will be called in a loop until any of the following:

1. The `n` number of steps have elapsed
2. The `until` function returns true
3. The exploration techniques `complete()` hooks (combined via the `SimulationManager.completion_mode` parameter/attribute - it is by default the `any` builtin function but can be changed to `all` for example) indicate that the analysis is complete
4. The stash being executed becomes empty

#### An aside: `explore()`

`SimulationManager.explore()` is a very thin wrapper around `run()` which adds the `Explorer` exploration technique, since performing one-off explorations is a very common action. Its code in its entirety is below:

```
num_find += len(self._stashes[find_stash]) if find_stash in self._stashes else 0
tech = self.use_technique(Explorer(find, avoid, find_stash, avoid_stash, cfg, num_find))

try:
    self.run(stash=stash, n=n, **kwargs)
finally:
    self.remove_technique(tech)

return self
```

### Exploration technique hooking

From here down, every function in the simulation manager can be instrumented by an exploration technique. The exact mechanism through which this works is that when you call `SimulationManager.use_technique()`, angr monkeypatches the simulation manager to replace any function implemented in the exploration technique's body with a function which will first call the exploration technique's function, and then on the second call will call the original function. This is somewhat messy to implement and certainly not thread safe by any means, but does produce a clean and powerful interface for exploration techniques to instrument stepping behavior, either before or after the original function is called, even choosing whether or not to call the original function whatsoever. Additionally, it allows multiple exploration techniques to hook the same function, as the monkeypatched function simply becomes the "original" function for the next-applied hook.

#### `step()`

There is a lot of complicated logic in `step()` to handle degenerate cases - mostly implementing the population of the `deadended` stash, the `save_unsat` option, and calling the `filter()` exploration technique hooks. Beyond this, though, most of the logic is looping through the stash specified by the `stash` argument and calling `step_state()` on each state, then applying the dict result of `step_state()` to the stash list. Finally, if the `step_func` parameter is provided, it is called with the simulation manager as a parameter before the step ends.

#### `step_state()`

The default `step_state()`, which can be overridden or instrumented by exploration techniques, is also simple - it calls `successors()`, which returns a `SimSuccessors` object, and then translates it into a dict mapping stash names to new states which should be added to that stash. It also implements error handling - if `successors()` throws an error, it will be caught and an `ErrorRecord` will be inserted into `SimulationManager.errored`.

#### successors()

We've almost made it out of SimulationManager. `successors()`, which can also be instrumented by exploration techniques, is supposed to take a state and step it forward, returning a `SimSuccessors` object categorizing its successors independently of any stash logic. If the `successor_func` parameter was provided, it is used and its return value is returned directly. If this parameter was not provided, we use the `project.factory.successors` method to tick the state forward and get our `SimSuccessors`.

### 5.2.2 The Engine

When we get to the actual successors generation, we need to figure out how to actually perform the execution. Hopefully, the angr documentation has been organized in a way such that by the time you reach this page, you know that a `SimEngine` is a device that knows how to take a state and produce its successors. There is only one "default engine" per project, but you can provide the `engine` parameter to specify which engine will be used to perform the step.

Keep in mind that this parameter can be provided way at the top, to `.step()`, `.explore()`, `.run()` or anything else that starts execution, and they will be filtered down to this level. Any additional parameters will continue being passed down, until they reach the part of the engine they are intended for. The engine will discard any parameters it doesn't understand.

Generally, the main entry point of an engine is `SimEngine.process()`, which can return whatever result it likes, but for simulation managers, engines are required to use `SuccessorsMixin`, which provides a `process()` method, which creates a `SimSuccessors` object and then calls `process_successors()` so that other mixins can fill it out.

angr's default engine, the `UberEngine`, contains several mixins which provide the `process_successors()` method:

- `SimEngineFailure` - handles stepping states with degenerate jumpkinds
- `SimEngineSyscall` - handles stepping states which have performed a syscall and need it executed
- `HooksMixin` - handles stepping states which have reached a hooked address and need the hook executed
- `SimEngineUnicorn` - executes machine code via the unicorn engine
- `SootMixin` - executes java bytecode via the SOOT IR
- `HeavyVEXMixin` - executes machine code via the VEX IR

Each of these mixins is implemented to fill out the `SimSuccessors` object if they can handle the current state, otherwise they call `super()` to pass the job on to the next class in the stack.

### 5.2.3 Engine mixins

`SimEngineFailure` handles error cases. It is only used when the previous jumpkind is one of `Ijk_EmFail`, `Ijk_MapFail`, `Ijk_Sig*`, `Ijk_NoDecode` (but only if the address is not hooked), or `Ijk_Exit`. In the first four cases, its action is to raise an exception. In the last case, its action is to simply produce no successors.

`SimEngineSyscall` services syscalls. It is used when the previous jumpkind is anything of the form `Ijk_Sys*`. It works by making a call into `SimOS` to retrieve the SimProcedure that should be run to respond to this syscall, and then running it! Pretty simple.

`HooksMixin` provides the hooking functionality in angr. It is used when a state is at an address that is hooked, and the previous jumpkind is *not* `Ijk_NoHook`. It simply looks up the associated SimProcedure and runs it on the state! It also takes the parameter `procedure`, which will cause the given procedure to be run for the current step even if the address is not hooked.

`SimEngineUnicorn` performs concrete execution with the Unicorn Engine. It is used when the state option `o.UNICORN` is enabled, and a myriad of other conditions designed for maximum efficiency (described below) are met.

`SootMixin` performs execution over the SOOT IR. Not very important unless you are analyzing java bytecode, in which case it is very important.

`SimEngineVEX` is the big fellow. It is used whenever any of the previous can't be used. It attempts to lift bytes from the current address into an IRSB, and then executes that IRSB symbolically. There are a huge number of parameters that can control this process, so it is best to reference the API doc for `angr.engines.vex.engine.SimEngineVEX.process()` describing them.

The exact process by which SimEngineVEX digs into an IRSB is a little complicated, but essentially it runs all the block's statements in order. This code is worth reading if you want to see the true inner core of angr's symbolic execution.

### 5.2.4 When using Unicorn Engine

If you add the `o.UNICORN` state option, at every step `SimEngineUnicorn` will be invoked, and try to see if it is allowed to use Unicorn to execute concretely.

What you REALLY want to do is to add the predefined set `o.unicorn` (lowercase) of options to your state:

```
unicorn = { UNICORN, UNICORN_SYM_REGS_SUPPORT, INITIALIZE_ZERO_REGISTERS, UNICORN_HANDLE_
↪TRANSMIT_SYSCALL }
```

These will enable some additional functionalities and defaults which will greatly enhance your experience. Additionally, there are a lot of options you can tune on the `state.unicorn` plugin.

A good way to understand how unicorn works is by examining the logging output (`logging.getLogger('angr.engines.unicorn_engine').setLevel('DEBUG'); logging.getLogger('angr.state_plugins.unicorn_engine').setLevel('DEBUG')` from a sample run of unicorn.

```
INFO    | 2017-02-25 08:19:48,012 | angr.state_plugins.unicorn | started emulation at␣
↪0x4012f9 (1000000 steps)
```

Here, angr diverts to unicorn engine, beginning with the basic block at 0x4012f9. The maximum step count is set to 1000000, so if execution stays in Unicorn for 1000000 blocks, it'll automatically pop out. This is to avoid hanging in an infinite loop. The block count is configurable via the `state.unicorn.max_steps` variable.

```
INFO    | 2017-02-25 08:19:48,014 | angr.state_plugins.unicorn | mmap [0x401000,␣
↪0x401fff], 5 (symbolic)
INFO    | 2017-02-25 08:19:48,016 | angr.state_plugins.unicorn | mmap [0x7fffffffffe0000,
↪ 0x7fffffffffeffff], 3 (symbolic)
INFO    | 2017-02-25 08:19:48,019 | angr.state_plugins.unicorn | mmap [0x6010000,␣
↪0x601ffff], 3
INFO    | 2017-02-25 08:19:48,022 | angr.state_plugins.unicorn | mmap [0x602000,␣
↪0x602fff], 3 (symbolic)
INFO    | 2017-02-25 08:19:48,023 | angr.state_plugins.unicorn | mmap [0x400000,␣
↪0x400fff], 5
INFO    | 2017-02-25 08:19:48,025 | angr.state_plugins.unicorn | mmap [0x7000000,␣
↪0x7000fff], 5
```

angr performs lazy mapping of data that is accessed by unicorn engine, as it is accessed. 0x401000 is the page of instructions that it is executing, 0x7fffffffffe0000 is the stack, and so on. Some of these pages are symbolic, meaning that they contain at least some data that, when accessed, will cause execution to abort out of Unicorn.

```
INFO    | 2017-02-25 08:19:48,037 | angr.state_plugins.unicorn | finished emulation at␣
↪0x7000080 after 3 steps: STOP_STOPPOINT
```

Execution stays in Unicorn for 3 basic blocks (a computational waste, considering the required setup), after which it reaches a simprocedure location and jumps out to execute the simproc in angr.

```
INFO    | 2017-02-25 08:19:48,076 | angr.state_plugins.unicorn | started emulation at␣
↪0x40175d (1000000 steps)
INFO    | 2017-02-25 08:19:48,077 | angr.state_plugins.unicorn | mmap [0x401000,␣
↪0x401fff], 5 (symbolic)
INFO    | 2017-02-25 08:19:48,079 | angr.state_plugins.unicorn | mmap [0x7fffffffffe0000,
↪ 0x7fffffffffeffff], 3 (symbolic)
INFO    | 2017-02-25 08:19:48,081 | angr.state_plugins.unicorn | mmap [0x6010000,␣
↪0x601ffff], 3
```

After the simprocedure, execution jumps back into Unicorn.

```
WARNING | 2017-02-25 08:19:48,082 | angr.state_plugins.unicorn | fetching empty page␣
↪[0x0, 0xfff]
INFO    | 2017-02-25 08:19:48,103 | angr.state_plugins.unicorn | finished emulation at␣
↪0x401777 after 1 steps: STOP_EXECNONE
```

Execution bounces out of Unicorn almost right away because the binary accessed the zero-page.

```
INFO    | 2017-02-25 08:19:48,120 | angr.engines.unicorn_engine | not enough runs since␣
↪last unicorn (100)
INFO    | 2017-02-25 08:19:48,125 | angr.engines.unicorn_engine | not enough runs since␣
↪last unicorn (99)
```

To avoid thrashing in and out of Unicorn (which is expensive), we have cooldowns (attributes of the `state.unicorn` plugin) that wait for certain conditions to hold (i.e., no symbolic memory accesses for X blocks) before jumping back into unicorn when a unicorn run is aborted due to anything but a simprocedure or syscall. Here, the condition it's waiting for is for 100 blocks to be executed before jumping back in.

## 5.3 What's Up With Mixins, Anyway?

If you are trying to work more intently with the deeper parts of angr, you will need to understand one of the design patterns we use frequently: the mixin pattern.

In brief, the mixin pattern is where Python's subclassing features is used not to implement IS-A relationships (a Child is a kind of Person) but instead to implement pieces of functionality for a type in different classes to make more modular and maintainable code. Here's an example of the mixin pattern in action:

```
class Base:
    def add_one(self, v):
        return v + 1

class StringsMixin(Base):
    def add_one(self, v):
        coerce = type(v) is str
        if coerce:
            v = int(v)
        result = super().add_one(v)
        if coerce:
            result = str(result)
        return result

class ArraysMixin(Base):
    def add_one(self, v):
```

(continues on next page)

(continued from previous page)

```
        if type(v) is list:
            return [super().add_one(v_x) for v_x in v]
        else:
            return super().add_one(v)

class FinalClass(ArraysMixin, StringsMixin, Base):
    pass
```

With this construction, we are able to define a very simple interface in the `Base` class, and by "mixing in" two mixins, we can create the `FinalClass` which has the same interface but with additional features. This is accomplished through Python's powerful multiple inheritance model, which handles method dispatch by creating a *method resolution order*, or MRO, which is unsurprisingly a list which determines the order in which methods are called as execution proceeds through `super()` calls. You can view a class' MRO as such:

```
FinalClass.__mro__

(FinalClass, ArraysMixin, StringsMixin, Base, object)
```

This means that when we take an instance of `FinalClass` and call `add_one()`, Python first checks to see if `FinalClass` defines an `add_one`, and then `ArraysMixin`, and so on and so forth. Furthermore, when `ArraysMixin` calls `super().add_one()`, Python will skip past `ArraysMixin` in the MRO, first checking if `StringsMixin` defines an `add_one`, and so forth.

Because multiple inheritance can create strange dependency graphs in the subclass relationship, there are rules for generating the MRO and for determining if a given mix of mixins is even allowed. This is important to understand when building complex classes with many mixins which have dependencies on each other. In short: left-to-right, depth-first, but deferring any base classes which are shared by multiple subclasses (the merge point of a diamond pattern in the inheritance graph) until the last point where they would be encountered in this depth-first search. For example, if you have classes A, B(A), C(B), D(A), E(C, D), then the method resolution order will be E, C, B, D, A. If there is any case in which the MRO would be ambiguous, the class construction is illegal and will throw an exception at import time.

This is complicated! If you find yourself confused, the canonical document explaining the rationale, history, and mechanics of Python's multiple inheritance can be found here.

### 5.3.1 Mixins in Claripy Solvers

**Todo**

Write this section

### 5.3.2 Mixins in angr Engines

The main entry point to a SimEngine is `process()`, but how do we determine what that does?

The mixin model is used in SimEngine and friends in order to allow pieces of functionality to be reused between static and symbolic analyses. The default engine, `UberEngine`, is defined as follows:

```
class UberEngine(SimEngineFailure,
   SimEngineSyscall,
   HooksMixin,
   SimEngineUnicorn,
   SuperFastpathMixin,
```

(continues on next page)

(continued from previous page)

```
    TrackActionsMixin,
    SimInspectMixin,
    HeavyResilienceMixin,
    SootMixin,
    HeavyVEXMixin
):
    pass
```

Each of these mixins provides either execution through a different medium or some additional instrumentation feature. Though they are not listed here explicitly, there are some base classes implicit to this hierarchy which set up the way this class is traversed. Most of these mixins inherit from `SuccessorsMixin`, which is what provides the basic `process()` implementation. This function sets up the `SimSuccessors` for the rest of the mixins to fill in, and then calls `process_successors()`, which each of the mixins which provide some mode of execution implement. If the mixin can handle the step, it does so and returns, otherwise it calls `super().process_successors()`. In this way, the MRO for the engine class determines what the order of precedence for the engine's pieces is.

### HeavyVEXMixin and friends

Let's take a closer look at the last mixin, `HeavyVEXMixin`. If you look at the module hierarchy of the angr `engines` submodule, you will see that the `vex` submodule has a lot of pieces in it which are organized by how tightly tied to particular state types or data types they are. The heavy VEX mixin is one version of the culmination of all of these. Let's look at its definition:

```
class HeavyVEXMixin(SuccessorsMixin, ClaripyDataMixin, SimStateStorageMixin, VEXMixin,␣
↪VEXLifter):
    ...
    # a WHOLE lot of implementation
```

So, the heavy VEX mixin is meant to provide fully instrumented symbolic execution on a SimState. What does this entail? The mixins tell the tale.

First, the plain `VEXMixin`. This mixin is designed to provide the barest-bones framework for processing a VEX block. Take a look at its source code. Its main purpose is to perform the preliminary digestion of the VEX IRSB and dispatch processing of it to methods which are provided by mixins - look at the methods which are either `pass` or `return NotImplemented`. Notice that absolutely none of its code makes any assumption whatsoever of what the type of `state` is or even what the type of the data words inside `state` are. This job is delegated to other mixins, making the `VEXMixin` an appropriate base class for literally any analysis on VEX blocks.

The next-most interesting mixin is the `ClaripyDataMixin`, whose source code is here. This mixin actually integrates the fact that we are executing over the domain of Claripy ASTs. It does this by implementing some of the methods which are unimplemented in the `VEXMixin`, most importantly the `ITE` expression, all the operations, and the clean helpers.

In terms of what it looks like to actually touch the SimState, the `SimStateStorageMixin` provides the glue between the `VEXMixin`'s interface for memory writes et al and SimState's interface for memory writes and such. It is unremarkable, except for a small interaction between it and the `ClaripyDataMixin`. The Claripy mixin also overrides the memory/register read/write functions, for the purpose of converting between the bitvector and floating-point types, since the vex interface expects to be able to load and store floats, but the SimState interface wants to load and store only bitvectors. Because of this, *the claripy mixin must come before the storage mixin in the MRO*. This is very much an interaction like the one in the add_one example at the start of this page - one mixin serves as a data filtering layer for another mixin.

#### Instrumenting the data layer

Let's turn our attention to a mixin which is not included in the `HeavyVEXMixin` but rather mixed into the `UberEngine` formula explicitly: the `TrackActionsMixin`. This mixin implements "SimActions", which is angr parlance for dataflow tracking. Again, look at the source code. The way it does this is that it *wraps and unwraps the data layer* to pass around additional information about data flows. Look at how it instruments `RdTmp`, for instance. It immediately `super()`-calls to the next method in the MRO, but instead of returning that data it returns a tuple of the data and its dependencies, which depending on whether you want temporary variables to be atoms in the dataflow model, will either be just the tmp which was read or the dependencies of the value written to that tmp.

This pattern continues for every single method that this mixin touches - any expression it receives must be unpacked into the expression and its dependencies, and any result must be packaged with its dependencies before it is returned. This works because the mixin above it makes no assumptions about what data it is passing around, and the mixin below it never gets to see any dependencies whatsoever. In fact, there could be multiple mixins performing this kind of wrap-unwrap trick and they could all coexist peacefully!

Note that a mixin which instruments the data layer in this way is *obligated* to override *every single method which takes or returns an expression value*, even if it doesn't perform any operation on the expression other than doing the wrapping and unwrapping. To understand why, imagine that the mixin does not override the `handle_vex_const` expression, so immediate value loads are not annotated with dependencies. The expression value which will be returned from the mixin which does provide `handle_vex_const` will not be a tuple of (expression, deps), it will just be the expression. Imagine this execution is taking place in the context of a `WrTmp(t0, Const(0))`. The const expression will be passed down to the `WrTmp` handler along with the identifier of the tmp to write to. However, since `handle_vex_stmt_WrTmp` *will* be overridden by our mixin which touches the data layer, it expects to be passed the tuple including the deps, and so it will crash when trying to unpack the not-a-tuple value.

In this way, you can sort of imagine that a mixin which instruments the data layer in this way is actually creating a contract within Python's nonexistent typesystem - you are guaranteed to receive back any types you return, but you must pass down any types you receive as return values from below.

### 5.3.3 Mixins in the memory model

**Todo**

write this section

## 5.4 Optimization considerations

The performance of angr as an analysis tool or emulator is greatly handicapped by the fact that lots of it is written in Python. Regardless, there are a lot of optimizations and tweaks you can use to make angr faster and lighter.

### 5.4.1 General speed tips

- *Use pypy*. Pypy is an alternate Python interpreter that performs optimized jitting of Python code. In our tests, it's a 10x speedup out of the box.
- *Only use the SimEngine mixins that you need*. SimEngine uses a mixin model which allows you to add and remove features by constructing new classes. The default engine mixes in every possible features, and the consequence of that is that it is slower than it needs to be. Look at the definition for `UberEngine` (the default SimEngine), copy its declaration, and remove all the base classes which provide features you don't need.
- *Don't load shared libraries unless you need them*. The default setting in angr is to try at all costs to find shared libraries that are compatible with the binary you've loaded, including loading them straight out of your OS libraries. This can complicate things in a lot of scenarios. If you're performing an analysis that's anything more abstract than bare-bones symbolic execution, ESPECIALLY control-flow graph construction, you might want

to make the tradeoff of sacrificing accuracy for tractability. angr does a reasonable job of making sane things happen when library calls to functions that don't exist try to happen.

- *Use hooking and SimProcedures*. If you're enabling shared libraries, then you definitely want to have SimProcedures written for any complicated library function you're jumping into. If there's no autonomy requirement for this project, you can often isolate individual problem spots where analysis hangs up and summarize them with a hook.
- *Use SimInspect*. *SimInspect* is the most underused and one of the most powerful features of angr. You can hook and modify almost any behavior of angr, including memory index resolution (which is often the slowest part of any angr analysis).
- *Write a concretization strategy*. A more powerful solution to the problem of memory index resolution is a concretization strategy.
- *Use the Replacement Solver*. You can enable it with the `angr.options.REPLACEMENT_SOLVER` state option. The replacement solver allows you to specify AST replacements that are applied at solve-time. If you add replacements so that all symbolic data is replaced with concrete data when it comes time to do the solve, the runtime is greatly reduced. The API for adding a replacement is `state.se._solver.add_replacement(old, new)`. The replacement solver is a bit finicky, so there are some gotchas, but it'll definitely help.

### 5.4.2 If you're performing lots of concrete or partially-concrete execution

- *Use the unicorn engine*. If you have unicorn engine installed, angr can be built to take advantage of it for concrete emulation. To enable it, add the options in the set `angr.options.unicorn` to your state. Keep in mind that while most items under `angr.options` are individual options, `angr.options.unicorn` is a bundle of options, and is thus a set. *NOTE*: At time of writing the official version of unicorn engine will not work with angr - we have a lot of patches to it to make it work well with angr. They're all pending pull requests at this time, so sit tight. If you're really impatient, ping us about uploading our fork!
- *Enable fast memory and fast registers*. The state options `angr.options.FAST_MEMORY` and `angr.options.FAST_REGISTERS` will do this. These will switch the memory/registers over to a less intensive memory model that sacrifices accuracy for speed. TODO: document the specific sacrifices. Should be safe for mostly concrete access though. NOTE: not compatible with concretization strategies.
- *Concretize your input ahead of time*. This is the approach taken by driller. When creating a state with `entry_state` or the like, you can create a SimFile filled with symbolic data, pass it to the initialization function as an argument `entry_state(..., stdin=my_simfile)`, and then constrain the symbolic data in the SimFile to what you want the input to be. If you don't require any tracking of the data coming from stdin, you can forego the symbolic part and just fill it with concrete data. If there are other sources of input besides standard input, do the same for those.
- *Use the afterburner*. While using unicorn, if you add the `UNICORN_THRESHOLD_CONCRETIZATION` state option, angr will accept thresholds after which it causes symbolic values to be concretized so that execution can spend more time in Unicorn. Specifically, the following thresholds exist:
  - `state.unicorn.concretization_threshold_memory` - this is the number of times a symbolic variable, stored in memory, is allowed to kick execution out of Unicorn before it is forcefully concretized and forced into Unicorn anyways.
  - `state.unicorn.concretization_threshold_registers` - this is the number of times a symbolic variable, stored in a register, is allowed to kick execution out of Unicorn before it is forcefully concretized and forced into Unicorn anyways.
  - `state.unicorn.concretization_threshold_instruction` - this is the number of times that any given instruction can force execution out of Unicorn (by running into symbolic data) before any symbolic data encountered at that instruction is concretized to force execution into Unicorn.

  You can get further control of what is and isn't concretized with the following sets:

- `state.unicorn.always_concretize` - a set of variable names that will always be concretized to force execution into unicorn (in fact, the memory and register thresholds just end up causing variables to be added to this list).
- `state.unicorn.never_concretize` - a set of variable names that will never be concretized and forced into Unicorn under any condition.
- `state.unicorn.concretize_at` - a set of instruction addresses at which data should be concretized and forced into Unicorn. The instruction threshold causes addresses to be added to this set.

Once something is concretized with the afterburner, you will lose track of that variable. The state will still be consistent, but you'll lose dependencies, as the stuff that comes out of Unicorn is just concrete bits with no memory of what variables they came from. Still, this might be worth it for the speed in some cases, if you know what you want to (or do not want to) concretize.

### 5.4.3 Memory optimization

The golden rule for memory optimization is to make sure you're not keeping any references to data you don't care about anymore, especially related to states which have been left behind. If you find yourself running out of memory during analysis, the first thing you want to do is make sure you haven't caused a state explosion, meaning that the analysis is accumulating program states too quickly. If the state count is in control, then you can start looking for reference leaks. A good tool to do this with is https://github.com/rhelmot/dumpsterdiver, which gives you an interactive prompt for exploring the reference graph of a Python process.

One specific consideration that should be made when analyzing programs with very long paths is that the state history is designed to accumulate data infinitely. This is less of a problem than it could be because the data is stored in a smart tree structure and never copied, but it will accumulate infinitely. To downsize a state's history and free all data related to old steps, call `state.history.trim()`.

One *particularly* problematic member of the history dataset is the basic block trace and the stack pointer trace. When using unicorn engine, these lists of ints can become huge very very quickly. To disable unicorn's capture of ip and sp data, remove the state options `UNICORN_TRACK_BBL_ADDRS` and `UNICORN_TRACK_STACK_POINTERS`.

## 5.5 Working with File System, Sockets, and Pipes

It's very important to be able to control the environment that emulated programs see, including how symbolic data is introduced from the environment! angr has a robust series of abstractions to help you set up the environment you want.

The root of any interaction with the filesystem, sockets, pipes, or terminals is a SimFile object. A SimFile is a *storage* abstraction that defines a sequence of bytes, symbolic or otherwise. There are several kinds of SimFiles which store their data very differently - the two easiest examples are `SimFile` (the base class is actually called `SimFileBase`), which stores files as a flat address-space of data, and `SimPackets`, which stores a sequence of variable-sized reads. The former is best for modeling programs that need to perform seeks on their files, and is the default storage for opened files, while the latter is best for modeling programs that depend on short-reads or use scanf, and is the default storage for stdin/stdout/stderr.

Because SimFiles can have such diverse storage mechanisms, the interface for interacting with them is *very* abstracted. You can read from the file from some position, you can write to the file at some position, you can ask how many bytes are currently stored in the file, and you can concretize the file, generating a testcase for it. If you know specifically which SimFile class you're working with, you can take much more powerful control over it, and as a result you're encouraged to manually create any files you want to work with when you create your initial state.

Specifically, each SimFile class creates its own abstraction of a "position" within the file - each read and write takes a position and returns a new position that you should use to continue from where you left off. If you're working with SimFiles of unknown type you have to treat this position as a totally opaque object with no semantics other than the contract with the read/write functions.

However! This is a very poor match to how programs generally interact with files, so angr also has a SimFileDescriptor abstraction, which provides the familiar read/write/seek/tell interfaces but will also return error conditions when the underlying storage don't support the appropriate operations - just like normal file descriptors!

You may access the mapping from file descriptor number to file descriptor object in `state.posix.fd`. See the API document for *angr.storage.file.SimFileDescriptorBase* for more details.

### 5.5.1 Just tell me how to do what I want to do!

Okay okay!!

To create a SimFile, you should just create an instance of the class you want to use. Refer to *angr.storage.file* for the full instructions.

Let's go through a few illustrative examples, which cover how you can work with a concrete file, a symbolic file, a file with mixed concrete and symbolic content, or streams.

#### Example 1: Create a file with concrete content

```
>>> import angr
>>> simfile = angr.SimFile('myconcretefile', content='hello world!\n')
```

Here's a nuance - you can't use SimFiles without a state attached, because reasons. You'll **never** have to do this in a real scenario (this operation happens automatically when you pass a SimFile into a constructor or the filesystem) but let's mock it up:

```
>>> proj = angr.Project('/bin/true')
>>> state = proj.factory.blank_state()
>>> simfile.set_state(state)
```

To demonstrate the behavior of these files we're going to use the fact that the default SimFile position is just the number of bytes from the start of the file. `SimFile.read` returns a tuple (bitvector data, actual size, new pos):

```
>>> data, actual_size, new_pos = simfile.read(0, 5)
>>> import claripy
>>> assert claripy.is_true(data == 'hello')
>>> assert claripy.is_true(actual_size == 5)
>>> assert claripy.is_true(new_pos == 5)
```

Continue the read, trying to read way too much:

```
>>> data, actual_size, new_pos = simfile.read(new_pos, 1000)
```

angr doesn't try to sanitize the data returned, only the size - we returned 1000 bytes! The intent is that you're only allowed to use up to actual_size of them.

```
>>> assert len(data) == 1000*8  # bitvector sizes are in bits
>>> assert claripy.is_true(actual_size == 8)
>>> assert claripy.is_true(data.get_bytes(0, 8) == ' world!\n')
>>> assert claripy.is_true(new_pos == 13)
```

#### Example 2: Create a file with symbolic content and a defined size

```
>>> simfile = angr.SimFile('mysymbolicfile', size=0x20)
>>> simfile.set_state(state)
```

(continues on next page)

(continued from previous page)

```
>>> data, actual_size, new_pos = simfile.read(0, 0x30)
>>> assert data.symbolic
>>> assert claripy.is_true(actual_size == 0x20)
```

The basic SimFile provides the same interface as `state.memory`, so you can load data directly:

```
>>> assert simfile.load(0, actual_size) is data.get_bytes(0, 0x20)
```

### Example 3: Create a file with constrained symbolic content

```
>>> bytes_list = [claripy.BVS('byte_%d' % i, 8) for i in range(32)]
>>> bytes_ast = claripy.Concat(*bytes_list)
>>> mystate = proj.factory.entry_state(stdin=angr.SimFile('/dev/stdin', content=bytes_
↪ast))
>>> for byte in bytes_list:
...     mystate.solver.add(byte >= 0x20)
...     mystate.solver.add(byte <= 0x7e)
```

### Example 4: Create a file with some mixed concrete and symbolic content, but no EOF

```
>>> variable = claripy.BVS('myvar', 10*8)
>>> simfile = angr.SimFile('mymixedfile', content=variable.concat(claripy.BVV('\n')),␣
↪has_end=False)
>>> simfile.set_state(state)
```

We can always query the number of bytes stored in the file:

```
>>> assert claripy.is_true(simfile.size == 11)
```

Reads will generate additional symbolic data past the current frontier:

```
>>> data, actual_size, new_pos = simfile.read(0, 15)
>>> assert claripy.is_true(actual_size == 15)
>>> assert claripy.is_true(new_pos == 15)

>>> assert claripy.is_true(data.get_bytes(0, 10) == variable)
>>> assert claripy.is_true(data.get_bytes(10, 1) == '\n')
>>> assert data.get_bytes(11, 4).symbolic
```

### Example 5: Create a file with a symbolic size (`has_end` is implicitly true here)

```
>>> symsize = claripy.BVS('mysize', 64)
>>> state.solver.add(symsize >= 10)
>>> state.solver.add(symsize < 20)
>>> simfile = angr.SimFile('mysymsizefile', size=symsize)
>>> simfile.set_state(state)
```

Reads will encode all possibilities:

```
>>> data, actual_size, new_pos = simfile.read(0, 30)
>>> assert set(state.solver.eval_upto(actual_size, 30)) == set(range(10, 20))
```

The maximum size can't be easily resolved, so the data returned is 30 bytes long, and we're supposed to use it conjunction with actual_size.

```
>>> assert len(data) == 30*8
```

Symbolic read sizes work too!

```
>>> symreadsize = claripy.BVS('myreadsize', 64)
>>> state.solver.add(symreadsize >= 5)
>>> state.solver.add(symreadsize < 30)
>>> data, actual_size, new_pos = simfile.read(0, symreadsize)
```

All sizes between 5 and 20 should be possible:

```
>>> assert set(state.solver.eval_upto(actual_size, 30)) == set(range(5, 20))
```

### Example 6: Working with streams (SimPackets)

So far, we've only used the SimFile class, which models a random-accessible file object. However, in real life, files are not everything. Streams (standard I/O, TCP, etc.) are a great example: While they hold data like a normal file does, they do not support random accesses, e.g., you cannot read out the second byte of stdin if you have already read passed that position, and you cannot modify any byte that has been previously sent out to a network endpoint. This allows us to design a simpler abstraction for streams in angr.

Believe it or not, this simpler abstraction for streams will benefit symbolic execution. Consider an example program that calls `scanf` N times to read in N strings. With a traditional SimFile, as we do not know the length of each input string, there does not exist any clear boundary in the file between these symbolic input strings. In this case, angr will perform N symbolic reads where each read will generate a gigantic tree of claripy ASTs, with string lengths being symbolic. This is a nightmare for constraint solving. Nevertheless, the fact that `scanf` is used on a stream (stdin) dictates that there will be zero overlap between individual reads, regardless of the sizes of each symbolic input string. We may as well model stdin as a stream that comprises of *consecutive packets*, instead of a file containing a sequence of bytes. Each of the packet can be of a fixed length or a symbolic length. Since there will be absolutely no byte overlap between packets, the constraints that angr will produce after executing this example program will be a lot simpler.

The key concept involved is "short reads", i.e. when you ask for `n` bytes but actually get back fewer bytes than that. We use a different class implementing SimFileBase, `SimPackets`, to automatically enable support for short reads. By default, stdin, stdout, and stderr are all SimPackets objects.

```
>>> simfile = angr.SimPackets('mypackets')
>>> simfile.set_state(state)
```

This'll just generate a single packet. For SimPackets, the position is just a packet number! If left unspecified, short_reads is determined from a state option.

```
>>> data, actual_size, new_pos = simfile.read(0, 20, short_reads=True)
>>> assert len(data) == 20*8
>>> assert set(state.solver.eval_upto(actual_size, 30)) == set(range(21))
```

Data in a SimPackets is stored as tuples of (packet data, packet size) in `.content`.

```
>>> print(simfile.content)
[(<BV160 packet_0_mypackets>, <BV64 packetsize_0_mypackets>)]

>>> simfile.read(0, 1, short_reads=False)
>>> print(simfile.content)
```

(continues on next page)

(continued from previous page)

```
[(<BV160 packet_0_mypackets>, <BV64 packetsize_0_mypackets>), (<BV8 packet_1_mypackets>,
↪<BV64 0x1>)]
```

So hopefully you understand sort of the kind of data that a SimFile can store and what'll happen when a program tries to interact with it with various combinations of symbolic and concrete data. Those examples only covered reads, but writes are pretty similar.

### 5.5.2 The filesystem, for real now

If you want to make a SimFile available to the program, we need to either stick it in the filesystem or serve stdin/stdout from it.

The simulated filesystem is the `state.fs` plugin. You can store, load, and delete files from the filesystem, with the `insert`, `get`, and `delete` methods. Refer to *angr.state_plugins.filesystem* for details.

So to make our file available as `/tmp/myfile`:

```
>>> state.fs.insert('/tmp/myfile', simfile)
>>> assert state.fs.get('/tmp/myfile') is simfile
```

Then, after execution, we would extract the file from the result state and use `simfile.concretize()` to generate a testcase to reach that state. Keep in mind that `concretize()` returns different types depending on the file type - for a SimFile it's a bytestring and for SimPackets it's a list of bytestrings.

The simulated filesystem supports a fun concept of "mounts", where you can designate a subtree as instrumented by a particular provider. The most common mount is to expose a part of the host filesystem to the guest, lazily importing file data when the program asks for it:

```
>>> state.fs.mount('/', angr.SimHostFilesystem('./guest_chroot'))
```

You can write whatever kind of mount you want to instrument filesystem access by subclassing `angr.SimMount`!

The filesystem also has a root directory, which `state.fs.chroot` changes - this is what the `chroot` simprocedure calls. The new root is resolved inside the filesystem the state already has, so a guest calling `chroot` can only reduce what it is able to reach; it never gains access to anything you did not mount yourself.

### 5.5.3 Stdio streams

For stdin and friends, it's a little more complicated. The relevant plugin is `state.posix`, which stores all abstractions relevant to a POSIX-compliant environment. You can always get a state's stdin SimFile with `state.posix.stdin`, but you can't just replace it - as soon as the state is created, references to this file are created in the file descriptors. Because of this you need to specify it at the time the POSIX plugin is created:

```
>>> state.register_plugin('posix', angr.state_plugins.posix.SimSystemPosix(stdin=simfile,
↪ stdout=simfile, stderr=simfile))
>>> assert state.posix.stdin is simfile
>>> assert state.posix.stdout is simfile
>>> assert state.posix.stderr is simfile
```

Or, there's a nice shortcut while creating the state if you only need to specify stdin:

```
>>> state = proj.factory.entry_state(stdin=simfile)
>>> assert state.posix.stdin is simfile
```

Any of those places you can specify a SimFileBase, you can also specify a string or a bitvector (a flat SimFile with fixed size will be created to hold it) or a SimFile type (it'll be instantiated for you).

## 5.6 Intermediate Representation

In order to be able to analyze and execute machine code from different CPU architectures, such as MIPS, ARM, and PowerPC in addition to the classic x86, angr performs most of its analysis on an *intermediate representation*, a structured description of the fundamental actions performed by each CPU instruction. By understanding angr's IR, VEX (which we borrowed from Valgrind), you will be able to write very quick static analyses and have a better understanding of how angr works.

The VEX IR abstracts away several architecture differences when dealing with different architectures, allowing a single analysis to be run on all of them:

- **Register names.** The quantity and names of registers differ between architectures, but modern CPU designs hold to a common theme: each CPU contains several general purpose registers, a register to hold the stack pointer, a set of registers to store condition flags, and so forth. The IR provides a consistent, abstracted interface to registers on different platforms. Specifically, VEX models the registers as a separate memory space, with integer offsets (e.g., AMD64's `rax` is stored starting at address 16 in this memory space).
- **Memory access.** Different architectures access memory in different ways. For example, ARM can access memory in both little-endian and big-endian modes. The IR abstracts away these differences.
- **Memory segmentation.** Some architectures, such as x86, support memory segmentation through the use of special segment registers. The IR understands such memory access mechanisms.
- **Instruction side-effects.** Most instructions have side-effects. For example, most operations in Thumb mode on ARM update the condition flags, and stack push/pop instructions update the stack pointer. Tracking these side-effects in an *ad hoc* manner in the analysis would be crazy, so the IR makes these effects explicit.

There are lots of choices for an IR. We use VEX, since the uplifting of binary code into VEX is quite well supported. VEX is an architecture-agnostic, side-effects-free representation of a number of target machine languages. It abstracts machine code into a representation designed to make program analysis easier. This representation has four main classes of objects:

- **Expressions.** IR Expressions represent a calculated or constant value. This includes memory loads, register reads, and results of arithmetic operations.
- **Operations.** IR Operations describe a *modification* of IR Expressions. This includes integer arithmetic, floating-point arithmetic, bit operations, and so forth. An IR Operation applied to IR Expressions yields an IR Expression as a result.
- **Temporary variables.** VEX uses temporary variables as internal registers: IR Expressions are stored in temporary variables between use. The content of a temporary variable can be retrieved using an IR Expression. These temporaries are numbered, starting at `t0`. These temporaries are strongly typed (e.g., "64-bit integer" or "32-bit float").
- **Statements.** IR Statements model changes in the state of the target machine, such as the effect of memory stores and register writes. IR Statements use IR Expressions for values they may need. For example, a memory store *IR Statement* uses an *IR Expression* for the target address of the write, and another *IR Expression* for the content.
- **Blocks.** An IR Block is a collection of IR Statements, representing an extended basic block (termed "IR Super Block" or "IRSB") in the target architecture. A block can have several exits. For conditional exits from the middle of a basic block, a special *Exit* IR Statement is used. An IR Expression is used to represent the target of the unconditional exit at the end of the block.

VEX IR is actually quite well documented in the `libvex_ir.h` file (https://github.com/angr/vex/blob/master/pub/libvex_ir.h) in the VEX repository. For the lazy, we'll detail some parts of VEX that you'll likely interact with fairly frequently. To begin with, here are some IR Expressions:

| IR Expression | Evaluated Value | VEX Output Example |
|---|---|---|
| Constant | A constant value. | 0x4:I32 |
| Read Temp | The value stored in a VEX temporary variable. | RdTmp(t10) |
| Get Register | The value stored in a register. | GET:I32(16) |
| Load Memory | The value stored at a memory address, with the address specified by another IR Expression. | LDle:I32 / LDbe:I64 |
| Operation | A result of a specified IR Operation, applied to specified IR Expression arguments. | Add32 |
| If-Then-Else | If a given IR Expression evaluates to 0, return one IR Expression. Otherwise, return another. | ITE |
| Helper Function | VEX uses C helper functions for certain operations, such as computing the conditional flags registers of certain architectures. These functions return IR Expressions. | function_name() |

These expressions are then, in turn, used in IR Statements. Here are some common ones:

| IR Statement | Meaning | VEX Output Example |
|---|---|---|
| Write Temp | Set a VEX temporary variable to the value of the given IR Expression. | WrTmp(t1) = (IR Expression) |
| Put Register | Update a register with the value of the given IR Expression. | PUT(16) = (IR Expression) |
| Store Memory | Update a location in memory, given as an IR Expression, with a value, also given as an IR Expression. | STle(0x1000) = (IR Expression) |
| Exit | A conditional exit from a basic block, with the jump target specified by an IR Expression. The condition is specified by an IR Expression. | if (condition) goto (Boring) 0x4000A00:I32 |

An example of an IR translation, on ARM, is produced below. In the example, the subtraction operation is translated into a single IR block comprising 5 IR Statements, each of which contains at least one IR Expression (although, in real life, an IR block would typically consist of more than one instruction). Register names are translated into numerical indices given to the *GET* Expression and *PUT* Statement. The astute reader will observe that the actual subtraction is modeled by the first 4 IR Statements of the block, and the incrementing of the program counter to point to the next instruction (which, in this case, is located at `0x59FC8`) is modeled by the last statement.

The following ARM instruction:

```
subs R2, R2, #8
```

Becomes this VEX IR:

```
t0 = GET:I32(16)
t1 = 0x8:I32
```

(continues on next page)

(continued from previous page)

```
t3 = Sub32(t0,t1)
PUT(16) = t3
PUT(68) = 0x59FC8:I32
```

Now that you understand VEX, you can actually play with some VEX in angr: We use a library called PyVEX that exposes VEX into Python. In addition, PyVEX implements its own pretty-printing so that it can show register names instead of register offsets in PUT and GET instructions.

PyVEX is accessible through angr through the `Project.factory.block` interface. There are many different representations you could use to access syntactic properties of a block of code, but they all have in common the trait of analyzing a particular sequence of bytes. Through the `factory.block` constructor, you get a `Block` object that can be easily turned into several different representations. Try `.vex` for a PyVEX IRSB, or `.capstone` for a Capstone block.

Let's play with PyVEX:

```
>>> import angr

# load the program binary
>>> proj = angr.Project("/bin/true")

# translate the starting basic block
>>> irsb = proj.factory.block(proj.entry).vex
# and then pretty-print it
>>> irsb.pp()

# translate and pretty-print a basic block starting at an address
>>> irsb = proj.factory.block(0x401340).vex
>>> irsb.pp()

# this is the IR Expression of the jump target of the unconditional exit at the end of␣
↪the basic block
>>> print(irsb.next)

# this is the type of the unconditional exit (e.g., a call, ret, syscall, etc)
>>> print(irsb.jumpkind)

# you can also pretty-print it
>>> irsb.next.pp()

# iterate through each statement and print all the statements
>>> for stmt in irsb.statements:
...     stmt.pp()

# pretty-print the IR expression representing the data, and the *type* of that IR␣
↪expression written by every store statement
>>> import pyvex
>>> for stmt in irsb.statements:
...     if isinstance(stmt, pyvex.IRStmt.Store):
...         print("Data:",)
...         stmt.data.pp()
...         print("")
...         print("Type:",)
...         print(stmt.data.result_type)
```

(continues on next page)

(continued from previous page)

```
...         print("")

# pretty-print the condition and jump target of every conditional exit from the basic␣
↪block
>>> for stmt in irsb.statements:
...     if isinstance(stmt, pyvex.IRStmt.Exit):
...         print("Condition:",)
...         stmt.guard.pp()
...         print("")
...         print("Target:",)
...         stmt.dst.pp()
...         print("")

# these are the types of every temp in the IRSB
>>> print(irsb.tyenv.types)

# here is one way to get the type of temp 0
>>> print(irsb.tyenv.types[0])
```

### 5.6.1 Condition flags computation (for x86 and ARM)

One of the most common instruction side-effects on x86 and ARM CPUs is updating condition flags, such as the zero flag, the carry flag, or the overflow flag. Computer architects usually put the concatenation of these flags (yes, concatenation of the flags, since each condition flag is 1 bit wide) into a special register (i.e. `EFLAGS/RFLAGS` on x86, `APSR/CPSR` on ARM). This special register stores important information about the program state, and is critical for correct emulation of the CPU.

VEX uses 4 registers as its "Flag thunk descriptors" to record details of the latest flag-setting operation. VEX has a lazy strategy to compute the flags: when an operation that would update the flags happens, instead of computing the flags, VEX stores a code representing this operation to the `cc_op` pseudo-register, and the arguments to the operation in `cc_dep1` and `cc_dep2`. Then, whenever VEX needs to get the actual flag values, it can figure out what the one bit corresponding to the flag in question actually is, based on its flag thunk descriptors. This is an optimization in the flags computation, as VEX can now just directly perform the relevant operation in the IR without bothering to compute and update the flags' value.

Amongst different operations that can be placed in `cc_op`, there is a special value 0 which corresponds to `OP_COPY` operation. This operation is supposed to copy the value in `cc_dep1` to the flags. It simply means that `cc_dep1` contains the flags' value. angr uses this fact to let us efficiently retrieve the flags' value: whenever we ask for the actual flags, angr computes their value, then dumps them back into `cc_dep1` and sets `cc_op = OP_COPY` in order to cache the computation. We can also use this operation to allow the user to write to the flags: we just set `cc_op = OP_COPY` to say that a new value being set to the flags, then set `cc_dep1` to that new value.

## 5.7 Working with Data and Conventions

Frequently, you'll want to access structured data from the program you're analyzing. angr has several features to make this less of a headache.

### 5.7.1 Working with types

angr has a system for representing types. These SimTypes are found in `angr.types` - an instance of any of these classes represents a type. Many of the types are incomplete unless they are supplemented with a SimState - their size depends on the architecture you're running under. You may do this with `ty.with_arch(arch)`, which returns a copy of itself, with the architecture specified.

angr also has a light wrapper around `pycparser`, which is a C parser. This helps with getting instances of type objects:

```
>>> import angr, monkeyhex

# note that SimType objects have their __repr__ defined to return their c type name,
# so this function actually returned a SimType instance.
>>> angr.types.parse_type('int')
int

>>> angr.types.parse_type('char **')
char**

>>> angr.types.parse_type('struct aa {int x; long y;}')
struct aa

>>> angr.types.parse_type('struct aa {int x; long y;}').fields
OrderedDict([('x', int), ('y', long)])
```

Additionally, you may parse C definitions and have them returned to you in a dict, either of variable/function declarations or of newly defined types:

```
>>> angr.types.parse_defns("int x; typedef struct llist { char* str; struct llist *next;␣
↪} list_node; list_node *y;")
{'x': int, 'y': struct llist*}

>>> defs = angr.types.parse_types("int x; typedef struct llist { char* str; struct llist␣
↪*next; } list_node; list_node *y;")
>>> defs
{'struct llist': struct llist, 'list_node': struct llist}

# if you want to get both of these dicts at once, use parse_file, which returns both in␣
↪a tuple.
>>> angr.types.parse_file("int x; typedef struct llist { char* str; struct llist *next; }
↪ list_node; list_node *y;")
({'x': int, 'y': struct llist*},
 {'struct llist': struct llist, 'list_node': struct llist})

>>> defs['list_node'].fields
OrderedDict([('str', char*), ('next', struct llist*)])

>>> defs['list_node'].fields['next'].pts_to.fields
OrderedDict([('str', char*), ('next', struct llist*)])

# If you want to get a function type and you don't want to construct it manually,
# you can use parse_type
>>> angr.types.parse_type("int (int y, double z)")
(int, double) -> int
```

And finally, you can register struct definitions for future use:

```
>>> angr.types.register_types(angr.types.parse_type('struct abcd { int x; int y; }'))
>>> angr.types.register_types(angr.types.parse_types('typedef long time_t;'))
>>> angr.types.parse_defns('struct abcd a; time_t b;')
{'a': struct abcd, 'b': long}
```

These type objects aren't all that useful on their own, but they can be passed to other parts of angr to specify data types.

### 5.7.2 Accessing typed data from memory

Now that you know how angr's type system works, you can unlock the full power of the `state.mem` interface! Any type that's registered with the types module can be used to extract data from memory.

```
>>> p = angr.Project('examples/fauxware/fauxware')
>>> s = p.factory.entry_state()
>>> s.mem[0x601048]
<<untyped> <unresolvable> at 0x601048>

>>> s.mem[0x601048].long
<long (64 bits) <BV64 0x4008d0> at 0x601048>

>>> s.mem[0x601048].long.resolved
<BV64 0x4008d0>

>>> s.mem[0x601048].long.concrete
0x4008d0

>>> s.mem[0x601048].struct.abcd
<struct abcd {
  .x = <BV32 0x4008d0>,
  .y = <BV32 0x0>
} at 0x601048>

>>> s.mem[0x601048].struct.abcd.x
<int (32 bits) <BV32 0x4008d0> at 0x601048>

>>> s.mem[0x601048].struct.abcd.y
<int (32 bits) <BV32 0x0> at 0x60104c>

>>> s.mem[0x601048].deref
<<untyped> <unresolvable> at 0x4008d0>

>>> s.mem[0x601048].deref.string
<string_t <BV64 0x534f534e45414b59> at 0x4008d0>

>>> s.mem[0x601048].deref.string.resolved
<BV64 0x534f534e45414b59>

>>> s.mem[0x601048].deref.string.concrete
b'SOSNEAKY'
```

The interface works like this:

- You first use [array index notation] to specify the address you'd like to load from
- If at that address is a pointer, you may access the `deref` property to return a SimMemView at the address present in memory.
- You then specify a type for the data by simply accessing a property of that name. For a list of supported types, look at `state.mem.types`.
- You can then *refine* the type. Any type may support any refinement it likes. Right now the only refinements

supported are that you may access any member of a struct by its member name, and you may index into a string or array to access that element.

- If the address you specified initially points to an array of that type, you can say `.array(n)` to view the data as an array of n elements.
- Finally, extract the structured data with `.resolved` or `.concrete`. `.resolved` will return bitvector values, while `.concrete` will return integer, string, array, etc values, whatever best represents the data.
- Alternately, you may store a value to memory, by assigning to the chain of properties that you've constructed. Note that because of the way Python works, `x = s.mem[...].prop; x = val` will NOT work, you must say `s.mem[...].prop = val`.

If you define a struct using `register_types(parse_type(struct_expr))`, you can access it here as a type:

```
>>> s.mem[p.entry].struct.abcd
<struct abcd {
  .x = <BV32 0x8949ed31>,
  .y = <BV32 0x89485ed1>
} at 0x400580>
```

### 5.7.3 Working with Calling Conventions

A calling convention is the specific means by which code passes arguments and return values through function calls. angr's abstraction of calling conventions is called SimCC. You can construct new SimCC instances through the angr object factory, with `p.factory.cc(...)`. This will give a calling convention which is guessed based your guest architecture and OS. If angr guesses wrong, you can explicitly pick one of the calling conventions in the `angr.calling_conventions` module.

If you have a very wacky calling convention, you can use `angr.calling_conventions.SimCCUsercall`. This will ask you to specify locations for the arguments and the return value. To do this, use instances of the `SimRegArg` or `SimStackArg` classes. You can find them in the factory - `p.factory.cc.Sim*Arg`.

Once you have a SimCC object, you can use it along with a SimState object and a function prototype (a SimTypeFunction) to extract or store function arguments more cleanly. Take a look at the `angr.calling_conventions.SimCC>` for details. Alternately, you can pass it to an interface that can use it to modify its own behavior, like `p.factory.call_state`, or...

### 5.7.4 Callables

Callables are a Foreign Functions Interface (FFI) for symbolic execution. Basic callable usage is to create one with `myfunc = p.factory.callable(addr)`, and then call it! `result = myfunc(args, ...)` When you call the callable, angr will set up a `call_state` at the given address, dump the given arguments into memory, and run a `path_group` based on this state until all the paths have exited from the function. Then, it merges all the result states together, pulls the return value out of that state, and returns it.

All the interaction with the state happens with the aid of a `SimCC` and a `SimTypeFunction`, to tell where to put the arguments and where to get the return value. It will try to use a sane default for the architecture, but if you'd like to customize it, you can pass a `SimCC` object in the `cc` keyword argument when constructing the callable. The `SimTypeFunction` is required - you must pass the `prototype` parameter. If you pass a string to this parameter it will be parsed as a function declaration.

You can pass symbolic data as function arguments, and everything will work fine. You can even pass more complicated data, like strings, lists, and structures as native Python data (use tuples for structures), and it'll be serialized as cleanly as possible into the state. If you'd like to specify a pointer to a certain value, you can wrap it in a `PointerWrapper` object, available as `p.factory.callable.PointerWrapper`. The exact semantics of how pointer-wrapping work are a little confusing, but they can be boiled down to "unless you specify it with a PointerWrapper or a specific SimArrayType, nothing will be wrapped in a pointer automatically unless it gets to the end and it hasn't yet been wrapped in a pointer

yet and the original type is a string, array, or tuple." The relevant code is actually in SimCC - it's the `setup_callsite` function.

If you don't care for the actual return value of the call, you can say `func.perform_call(arg, ...)`, and then the properties `func.result_state` and `func.result_path_group` will be populated. They will actually be populated even if you call the callable normally, but you probably care about them more in this case!

## 5.8 Solver Engine

angr's solver engine is called Claripy. Claripy exposes the following design:

- Claripy ASTs (the subclasses of claripy.ast.Base) provide a unified way to interact with concrete and symbolic expressions
- `Frontend`s provide different paradigms for evaluating these expressions. For example, the `FullFrontend` solves expressions using something like an SMT solver backend, while `LightFrontend` handles them by using an abstract (and approximating) data domain backend.
- Each `Frontend` needs to, at some point, do actual operation and evaluations on an AST. ASTs don't support this on their own. Instead, `Backend`s translate ASTs into backend objects (i.e., Python primitives for `BackendConcrete`, Z3 expressions for `BackendZ3`, strided intervals for `BackendVSA`, etc) and handle any appropriate state-tracking objects (such as tracking the solver state in the case of `BackendZ3`). Roughly speaking, frontends take ASTs as inputs and use backends to `backend.convert()` those ASTs into backend objects that can be evaluated and otherwise reasoned about.
- `FrontendMixin`s customize the operation of `Frontend`s. For example, `ModelCacheMixin` caches solutions from an SMT solver.
- The combination of a Frontend, a number of FrontendMixins, and a number of Backends comprise a claripy `Solver`.

Internally, Claripy seamlessly mediates the co-operation of multiple disparate backends – concrete bitvectors, VSA constructs, and SAT solvers. It is pretty badass.

Most users of angr will not need to interact directly with Claripy (except for, maybe, claripy AST objects, which represent symbolic expressions) – angr handles most interactions with Claripy internally. However, for dealing with expressions, an understanding of Claripy might be useful.

### 5.8.1 Claripy ASTs

Claripy ASTs abstract away the differences between mathematical constructs that Claripy supports. They define a tree of operations (i.e., `(a + b) / c`) on any type of underlying data. Claripy handles the application of these operations on the underlying objects themselves by dispatching requests to the backends.

Currently, Claripy supports the following types of ASTs:

| Nam | Description | Supported By (Claripy Backends) | Example Code |
|---|---|---|---|
| BV | This is a bitvector, whether symbolic (with a name) or concrete (with a value). It has a size (in bits). | BackendConcrete, BackendVSA, BackendZ3 | Create a 32-bit symbolic bitvector "x": *claripy.BVS('x', 32)* Create a 32-bit bitvector with the value *0xc001b3475*: *claripy.BVV(0xc001b3a75, 32)`</li><li>Create a 32-bit "strided interval" (see VSA documentation) that can be any divisible-by-10 number between 1000 and 2000: `claripy.SI(name='x', bits=32, lower_bound=1000, upper_bound=2000, stride=10)`</li></ul>* |
| FP | This is a floating-point number, whether symbolic (with a name) or concrete (with a value). | BackendConcrete, BackendZ3 | Create a `claripy.fp.FSORT_DOUBLE` symbolic floating point "b": `claripy.FPS('b', claripy.fp.FSORT_DOUBLE)`. Create a `claripy.fp.FSORT_FLOAT` floating point with value `3.2`: `claripy.FPV(3.2, claripy.fp.FSORT_FLOAT)`. |
| Bool | This is a boolean operation (True or False). | BackendConcrete, BackendVSA, BackendZ3 | `claripy.BoolV(True)`, or `claripy.true` or `claripy.false`, or by comparing two ASTs (i.e., `claripy.BVS('x', 32) < claripy.BVS('y', 32)` |

All of the above creation code returns claripy.AST objects, on which operations can then be carried out.

ASTs provide several useful operations.

```
>>> import claripy

>>> bv = claripy.BVV(0x41424344, 32)

# Size - you can get the size of an AST with .size()
>>> assert bv.size() == 32

# Reversing - .reversed is the reversed version of the BVV
>>> assert bv.reversed is claripy.BVV(0x44434241, 32)
>>> assert bv.reversed.reversed is bv

# Depth - you can get the depth of the AST
>>> print(bv.depth)
>>> assert bv.depth == 1
>>> x = claripy.BVS('x', 32)
>>> assert (x+bv).depth == 2
>>> assert ((x+bv)/10).depth == 3
```

Applying a condition (==, !=, etc) on ASTs will return an AST that represents the condition being carried out. For example:

```
>>> r = bv == x
>>> assert isinstance(r, claripy.ast.Bool)

>>> p = bv == bv
>>> assert isinstance(p, claripy.ast.Bool)
>>> assert p.is_true()
```

You can combine these conditions in different ways.

```
>>> q = claripy.And(claripy.Or(bv == x, bv * 2 == x, bv * 3 == x), x == 0)
>>> assert isinstance(p, claripy.ast.Bool)
```

The usefulness of this will become apparent when we discuss Claripy solvers.

In general, Claripy supports all of the normal Python operations (+, -, |, ==, etc), and provides additional ones via the Claripy instance object. Here's a list of available operations from the latter.

| Name | Description | Example |
|---|---|---|
| LShR | Logically shifts a bit expression (BVV, BV, SI) to the right. | `claripy.LShR(x, 10)` |
| SignExt | Sign-extends a bit expression. | `claripy.SignExt(32, x)` or `x.sign_extend(32)` |
| ZeroExt | Zero-extends a bit expression. | `claripy.ZeroExt(32, x)` or `x.zero_extend(32)` |
| Extract | Extracts the given bits (zero-indexed from the *right*, inclusive) from a bit expression. | Extract the rightmost byte of x: `claripy.Extract(7, 0, x)` or `x[7:0]` |
| Concat | Concatenates several bit expressions together into a new bit expression. | `claripy.Concat(x, y, z)` |
| RotateLeft | Rotates a bit expression left. | `claripy.RotateLeft(x, 8)` |
| RotateRight | Rotates a bit expression right. | `claripy.RotateRight(x, 8)` |
| Reverse | Endian-reverses a bit expression. | `claripy.Reverse(x)` or `x.reversed` |
| And | Logical And (on boolean expressions) | `claripy.And(x == y, x > 0)` |
| Or | Logical Or (on boolean expressions) | `claripy.Or(x == y, y < 10)` |
| Not | Logical Not (on a boolean expression) | `claripy.Not(x == y)` is the same as `x != y` |
| If | An If-then-else | Choose the maximum of two expressions: `claripy.If(x > y, x, y)` |
| ULE | Unsigned less than or equal to. | Check if x is less than or equal to y: `claripy.ULE(x, y)` |
| ULT | Unsigned less than. | Check if x is less than y: `claripy.ULT(x, y)` |
| UGE | Unsigned greater than or equal to. | Check if x is greater than or equal to y: `claripy.UGE(x, y)` |
| UGT | Unsigned greater than. | Check if x is greater than y: `claripy.UGT(x, y)` |
| SLE | Signed less than or equal to. | Check if x is less than or equal to y: `claripy.SLE(x, y)` |
| SLT | Signed less than. | Check if x is less than y: `claripy.SLT(x, y)` |
| SGE | Signed greater than or equal to. | Check if x is greater than or equal to y: `claripy.SGE(x, y)` |
| SGT | Signed greater than. | Check if x is greater than y: `claripy.SGT(x, y)` |

> **Note**
>
> The default Python >, <, >=, and <= are unsigned in Claripy. This is different than their behavior in Z3, because it seems more natural in binary analysis.

### 5.8.2 Solvers

The main point of interaction with Claripy are the Claripy Solvers. Solvers expose an API to interpret ASTs in different ways and return usable values. There are several different solvers.

| Name | Description |
|---|---|
| Solver | This is analogous to a `z3.Solver()`. It is a solver that tracks constraints on symbolic variables and uses a constraint solver (currently, Z3) to evaluate symbolic expressions. |
| SolverVSA | This solver uses VSA to reason about values. It is an *approximating* solver, but produces values without performing actual constraint solves. |
| SolverReplacement | This solver acts as a pass-through to a child solver, allowing the replacement of expressions on-the-fly. It is used as a helper by other solvers and can be used directly to implement exotic analyses. |
| SolverHybrid | This solver combines the SolverReplacement and the Solver (VSA and Z3) to allow for *approximating* values. You can specify whether or not you want an exact result from your evaluations, and this solver does the rest. |
| SolverComposite | This solver implements optimizations that solve smaller sets of constraints to speed up constraint solving. |

Some examples of solver usage:

```
# create the solver and an expression
>>> s = claripy.Solver()
>>> x = claripy.BVS('x', 8)

# now let's add a constraint on x
>>> s.add(claripy.ULT(x, 5))

>>> assert sorted(s.eval(x, 10)) == [0, 1, 2, 3, 4]
>>> assert s.max(x) == 4
>>> assert s.min(x) == 0

# we can also get the values of complex expressions
>>> y = claripy.BVV(65, 8)
>>> z = claripy.If(x == 1, x, y)
>>> assert sorted(s.eval(z, 10)) == [1, 65]

# and, of course, we can add constraints on complex expressions
>>> s.add(z % 5 != 0)
>>> assert s.eval(z, 10) == (1,)
>>> assert s.eval(x, 10) == (1,) # interestingly enough, since z can't be y, x can only␣
↪be 1!
```

Custom solvers can be built by combining a Claripy Frontend (the class that handles the actual interaction with SMT solver or the underlying data domain) and some combination of frontend mixins (that handle things like caching, filtering out duplicate constraints, doing opportunistic simplification, and so on).

### 5.8.3 Claripy Backends

Backends are Claripy's workhorses. Claripy exposes ASTs to the world, but when actual computation has to be done, it pushes those ASTs into objects that can be handled by the backends themselves. This provides a unified interface to the outside world while allowing Claripy to support different types of computation. For example, BackendConcrete provides computation support for concrete bitvectors and booleans, BackendVSA introduces VSA constructs such as StridedIntervals (and details what happens when operations are performed on them, and BackendZ3 provides support for symbolic variables and constraint solving.

There are a set of functions that a backend is expected to implement. For all of these functions, the "public" version is expected to be able to deal with claripy's AST objects, while the "private" version should only deal with objects specific to the backend itself. This is distinguished with Python idioms: a public function will be named func() while a private function will be _func(). All functions should return objects that are usable by the backend in its private methods. If this can't be done (i.e., some functionality is being attempted that the backend can't handle), the backend should raise a BackendError. In this case, Claripy will move on to the next backend in its list.

All backends must implement a `convert()` function. This function receives a claripy AST and should return an object that the backend can handle in its private methods. Backends should also implement a `convert()` method, which will receive anything that is *not* a claripy AST object (i.e., an integer or an object from a different backend). If `convert()` or `convert()` receives something that the backend can't translate to a format that is usable internally, the backend should raise BackendError, and thus won't be used for that object. All backends must also implement any functions of the base `Backend` abstract class that currently raise `NotImplementedError()`.

Claripy's contract with its backends is as follows: backends should be able to handle, in their private functions, any object that they return from their private *or* public functions. Claripy will never pass an object to any backend private function that did not originate as a return value from a private or public function of that backend. One exception to this is `convert()` and `convert()`, as Claripy can try to stuff anything it feels like into _convert() to see if the backend can handle that type of object.

#### Backend Objects

To perform actual, useful computation on ASTs, Claripy uses backend objects. A `BackendObject` is a result of the operation represented by the AST. Claripy expects these objects to be returned from their respective backends, and will pass such objects into that backend's other functions.

## 5.9 Symbolic memory addressing

angr supports *symbolic memory addressing*, meaning that offsets into memory may be symbolic. Our implementation of this is inspired by "Mayhem". Specifically, this means that angr concretizes symbolic addresses when they are used as the target of a write. This causes some surprises, as users tend to expect symbolic writes to be treated purely symbolically, or "as symbolically" as we treat symbolic reads, but that is not the default behavior. However, like most things in angr, this is configurable.

The address resolution behavior is governed by *concretization strategies*, which are subclasses of `angr.concretization_strategies.SimConcretizationStrategy`. Concretization strategies for reads are set in `state.memory.read_strategies` and for writes in `state.memory.write_strategies`. These strategies are called, in order, until one of them is able to resolve addresses for the symbolic index. By setting your own concretization strategies (or through the use of SimInspect `address_concretization` breakpoints, described above), you can change the way angr resolves symbolic addresses.

For example, angr's default concretization strategies for writes are:

1. A conditional concretization strategy that allows symbolic writes (with a maximum range of 128 possible solutions) for any indices that are annotated with `angr.plugins.symbolic_memory.MultiwriteAnnotation`.
2. A concretization strategy that simply selects the maximum possible solution of the symbolic index.

To enable symbolic writes for all indices, you can either add the `SYMBOLIC_WRITE_ADDRESSES` state option at state creation time or manually insert a `angr.concretization_strategies.SimConcretizationStrategyRange` object into `state.memory.write_strategies`. The strategy object takes a single argument, which is the maximum range of possible solutions that it allows before giving up and moving on to the next (presumably non-symbolic) strategy.

### 5.9.1 Writing concretization strategies

**Todo**

Write this section

## 5.10 Java Support

`angr` also supports symbolically executing Java code and Android apps! This also includes Android apps using a combination of compiled Java and native (C/C++) code.

**Warning**

Java support is experimental! Contribution from the community is highly encouraged! Pull requests are very welcomed!

We implemented Java support by lifting the compiled Java code, both Java and DEX bytecode, leveraging our Soot Python wrapper: pysoot. `pysoot` extracts a fully serializable interface from Android apps and Java code (unfortunately, as of now, it only works on Linux). For every class of the generated IR (for instance, `SootMethod`), you can nicely print its instructions (in a format similar to `Soot shimple`) using `print()` or `str()`.

**Note**

Windows and macOS support is available on a branch. It should pass all tests and generally work well, but due to issues integrating JPype into CI infrastructure, it has not yet been merged.

We then leverage the generated IR in a new angr engine able to run code in Soot IR: angr/engines/soot/engine.py. This engine is also able to automatically switch to executing native code if the Java code calls any native method using the JNI interface.

Together with the symbolic execution, we also implemented some basic static analysis, specifically a basic CFG reconstruction analysis. Moreover, we added support for string constraint solving, modifying claripy and using the CVC4 solver.

### 5.10.1 How to install

Java support requires the `pysoot` package, which is not included in the default angr installation. You can install it from GitHub using pip:

```
git clone https://github.com/angr/pysoot.git
cd pysoot
pip install -e .
```

Alternatively, pysoot can be installed with the setup script in angr-dev:

```
./setup.sh pysoot
```

#### Analyzing Android apps.

Analyzing Android apps (`.APK` files, containing Java code compiled to the `DEX` format) requires the Android SDK. Typically, it is installed in `<HOME>/Android/SDK/platforms/platform-XX/android.jar`, where `XX` is the Android SDK version used by the app you want to analyze (you may want to install all the platforms required by the Android apps you want to analyze).

### 5.10.2 Examples

There are multiple examples available:

- Easy Java crackmes: java_crackme1, java_simple3, java_simple4
- A more complex example (solving a CTF challenge): ictf2017_javaisnotfun, blogpost
- Symbolically executing an Android app (using a mix of Java and native code): java_androidnative1
- Many other low-level tests: test_java

## 5.11 Debug variable resolution

angr now support resolve source level variable (debug variable) in binary with debug information. This article will introduce you how to use it.

### 5.11.1 Setting up

To use it you need binary that is compiled with dwarf debugging information (ex: `gcc -g`) and load in angr with the option `load_debug_info`. After that you need to run `project.kb.dvars.load_from_dwarf()` to set up the feature and we're set.

Overall it looks like this:

```
# compile your binary with debug information
gcc -g -o debug_var debug_var.c
```

```
>>> import angr
>>> project = angr.Project('./examples/debug_var/simple_var', load_debug_info = True)
>>> project.kb.dvars.load_from_dwarf()
```

### 5.11.2 Core feature

With things now set up you can view the value in the angr memory view of the debug variable within a state with: `state.dvars['variable_name'].mem` or the value that it point to if it is a pointer with: `state.dvars['pointer_name'].deref.mem`. Here are some example:

Given the source code in `examples/debug_var/simple_var.c`

```
#include<stdio.h>

int global_var = 100;
int main(void){
   int a = 10;
   int* b = &a;
```

(continues on next page)

(continued from previous page)

```
   printf("%d\n", *b);
   {
      int a = 24;
      *b = *b + a;
      int c[] = {5, 6, 7, 8};
      printf("%d\n", a);
   }
   return 0;
}
```

```
# Get a state before executing printf(%d\n", *b) (line 7)
# the addr to line 7 is 0x401193 you can search for it with
>>> project.loader.main_object.addr_to_line
{...}
>>> addr = 0x401193
# Create an simulation manager and run to that addr
>>> simgr = project.factory.simgr()
>>> simgr.explore(find = addr)
<SimulationManager with 1 found>
>>> state = simgr.found[0]
# Resolve 'a' in state
>>> state.dvars['a'].mem
<int (32 bits) <BV32 0xa> at 0x7fffffffffeff30>
# Dereference pointer b
>>> state.dvars['b'].deref.mem
<int (32 bits) <BV32 0xa> at 0x7fffffffffeff30>
# It works as expected when resolving the value of b gives the address of a
>>> state.dvars['b'].mem
<reg64_t <BV64 0x7fffffffffeff30> at 0x7fffffffffeff38>
```

Side-note: For string type you can use `.string` instead of `.mem` to resolve it. For struct type you can resolve its member by `.member("member_name").mem`. For array type you can use `.array(index).mem` to access the element in array.

## 5.12 Variable visibility

If you have many variable with the same name but in different scope, calling `state.dvars['var_name']` would resolve the variable with the nearest scope.

Example:

```
# Find the addr before executing printf("%d\n", a) (line 12)
# with the same method to find addr
>>> addr = 0x4011e0
# Explore until find state
>>> simgr.move(from_stash='found', to_stash='active')
<SimulationManager with 1 active>
>>> simgr.explore(find = addr)
<SimulationManager with 1 found>
>>> state = simgr.found[0]
# Resolve 'a' in state before execute line 10
>>> state.dvars['a'].mem
<int (32 bits) <BV32 0x18> at 0x7fffffffffeff34>
```

Congratulation, you've now know how to resolve debug variable using angr, for more info check out the api-doc.

CHAPTER

SIX

# EXTENDING ANGR

## 6.1 Hooks and SimProcedures

Hooks in angr are very powerful! You can use them to modify a program's behavior in any way you could imagine. However, the exact way you might want to program a specific hook may be non-obvious. This chapter should serve as a guide when programming SimProcedures.

### 6.1.1 Quick Start

Here's an example that will remove all bugs from any program:

```
>>> from angr import Project, SimProcedure
>>> project = Project('examples/fauxware/fauxware')

>>> class BugFree(SimProcedure):
...     def run(self, argc, argv):
...         print('Program running with argc=%s and argv=%s' % (argc, argv))
...         return 0

# this assumes we have symbols for the binary
>>> project.hook_symbol('main', BugFree())

# Run a quick execution!
>>> simgr = project.factory.simulation_manager()
>>> simgr.run()  # step until no more active states
Program running with argc=<SAO <BV64 0x0>> and argv=<SAO <BV64 0x7fffffffffeffa0>>
<SimulationManager with 1 deadended>
```

Now, whenever program execution reaches the main function, instead of executing the actual main function, it will execute this procedure! It just prints out a message, and returns.

Now, let's talk about what happens on the edge of this function! When entering the function, where do the values that go into the arguments come from? You can define your `run()` function with however many arguments you like, and the SimProcedure runtime will automatically extract from the program state those arguments for you, via a *calling convention*, and call your run function with them. Similarly, when you return a value from the run function, it is placed into the state (again, according to the calling convention), and the actual control-flow action of returning from a function is performed, which depending on the architecture may involve jumping to the link register or jumping to the result of a stack pop.

It should be clear at this point that the SimProcedure we just wrote is meant to totally replace whatever function it is hooked over top of. In fact, the original use case for SimProcedures was replacing library functions. More on that later.

### 6.1.2 Implementation Context

On a `Project` class, the dict `project._sim_procedures` is a mapping from address to `SimProcedure` instances. When the *execution pipeline* reaches an address that is present in that dict, that is, an address that is hooked, it will execute `project._sim_procedures[address].execute(state)`. This will consult the calling convention to extract the arguments, make a copy of itself in order to preserve thread safety, and run the `run()` method. It is important to produce a new instance of the SimProcedure for each time it is run, since the process of running a SimProcedure necessarily involves mutating state on the SimProcedure instance, so we need separate ones for each step, lest we run into race conditions in multithreaded environments.

#### kwargs

This hierarchy implies that you might want to reuse a single SimProcedure in multiple hooks. What if you want to hook the same SimProcedure in several places, but tweaked slightly each time? angr's support for this is that any additional keyword arguments you pass to the constructor of your SimProcedure will end up getting passed as keyword args to your SimProcedure's `run()` method. Pretty cool!

### 6.1.3 Data Types

If you were paying attention to the example earlier, you noticed that when we printed out the arguments to the `run()` function, they came out as a weird `<SAO <BV64 0xSTUFF>>` class. This is a `SimActionObject`. Basically, you don't need to worry about it too much, it's just a thin wrapper over a normal bitvector. It does a bit of tracking of what exactly you do with it inside the SimProcedure—this is helpful for static analysis.

You may also have noticed that we directly returned the Python int `0` from the procedure. This will automatically be promoted to a word-sized bitvector! You can return a native number, a bitvector, or a SimActionObject.

When you want to write a procedure that deals with floating point numbers, you will need to specify the calling convention manually. It's not too hard, just provide a cc to the hook: `` `cc = project.factory.cc_from_arg_kinds((True, True), ret_fp=True) `` and `project.hook(address, ProcedureClass(cc=mycc))` This method for passing in a calling convention works for all calling conventions, so if angr's autodetected one isn't right, you can fix that.

### 6.1.4 Control Flow

How can you exit a SimProcedure? We've already gone over the simplest way to do this, returning a value from `run()`. This is actually shorthand for calling `self.ret(value)`. `self.ret()` is the function which knows how to perform the specific action of returning from a function.

SimProcedures can use lots of different functions like this!

- `ret(expr)`: Return from a function
- `jump(addr)`: Jump to an address in the binary
- `exit(code)`: Terminate the program
- `call(addr, args, continue_at)`: Call a function in the binary
- `inline_call(procedure, *args)`: Call another SimProcedure in-line and return the results

That second-last one deserves some looking-at. We'll get there after a quick detour...

#### Conditional Exits

What if we want to add a conditional branch out of a SimProcedure? In order to do that, you'll need to work directly with the SimSuccessors object for the current execution step.

The interface for this is `self.successors.add_successor(state, addr, guard, jumpkind)`. All of these parameters should have an obvious meaning if you've followed along so far. Keep in mind that the state you pass in will NOT be copied and WILL be mutated, so be sure to make a copy beforehand if there will be more work to do!

### SimProcedure Continuations

How can we call a function in the binary and have execution resume within our SimProcedure? There is a whole bunch of infrastructure called the "SimProcedure Continuation" that will let you do this. When you use `self.call(addr, args, continue_at)`, `addr` is expected to be the address you'd like to call, `args` is the tuple of arguments you'd like to call it with, and `continue_at` is the name of another method in your SimProcedure class that you'd like execution to continue at when it returns. This method must have the same signature as the `run()` method. Furthermore, you can pass the keyword argument `cc` as the calling convention that ought to be used to communicate with the callee.

When you do this, you finish your current step, and execution will start again at the next step at the function you've specified. When that function returns, it has to return to some concrete address! That address is specified by the SimProcedure runtime: an address is allocated in angr's externs segment to be used as the return site for returning to the given method call. It is then hooked with a copy of the procedure instance tweaked to run the specified `continue_at` function instead of `run()`, with the same args and kwargs as the first time.

There are two pieces of metadata you need to attach to your SimProcedure class in order to use the continuation subsystem correctly:

- Set the class variable `IS_FUNCTION = True`
- Set the class variable `local_vars` to a tuple of strings, where each string is the name of an instance variable on your SimProcedure whose value you would like to persist to when you return. Local variables can be any type so long as you don't mutate their instances.

You may have guessed by now that there exists some sort of auxiliary storage in order to hold on to all this data. You would be right! The state plugin `state.callstack` has an entry called `.procedure_data` which is used by the SimProcedure runtime to store information local to the current call frame. angr tracks the stack pointer in order to make the current top of the `state.callstack` a meaningful local data store. It's stuff that ought to be stored in memory in a stack frame, but the data can't be serialized and/or memory allocation is hard.

As an example, let's look at the SimProcedure that angr uses internally to run all the shared library initializers for a `full_init_state` for a linux program:

```
class LinuxLoader(angr.SimProcedure):
    NO_RET = True
    IS_FUNCTION = True
    local_vars = ('initializers',)

    def run(self):
        self.initializers = self.project.loader.initializers
        self.run_initializer()

    def run_initializer(self):
        if len(self.initializers) == 0:
            self.project._simos.set_entry_register_values(self.state)
            self.jump(self.project.entry)
        else:
            addr = self.initializers[0]
            self.initializers = self.initializers[1:]
            self.call(addr, (self.state.posix.argc, self.state.posix.argv, self.state.
↪posix.environ), 'run_initializer')
```

This is a particularly clever usage of the SimProcedure continuations. First, notice that the current project is available for use on the procedure instance. This is some powerful stuff you can get yourself into; for safety you generally only want

to use the project as a read-only or append-only data structure. Here we're just getting the list of dynamic initializers from the loader. Then, for as long as the list isn't empty, we pop a single function pointer out of the list, being careful not to mutate the list, since the list object is shared across states, and then call it, returning to the `run_initializer` function again. When we run out of initializers, we set up the entry state and jump to the program entry point.

Very cool!

### 6.1.5 Global Variables

As a brief aside, you can store global variables in `state.globals`. This is a dictionary that just gets shallow-copied from state to successor state. Because it's only a shallow copy, its members are the same instances, so the same rules as local variables in SimProcedure continuations apply. You need to be careful not to mutate any item that is used as a global variable unless you know exactly what you're doing.

### 6.1.6 Helping out static analysis

We've already looked at the class variable `IS_FUNCTION`, which allows you to use the SimProcedure continuation. There are a few more class variables you can set, though these ones have no direct benefit to you - they merely mark attributes of your function so that static analysis knows what it's doing.

- `NO_RET`: Set this to true if control flow will never return from this function
- `ADDS_EXITS`: Set this to true if you do any control flow other than returning
- `IS_SYSCALL`: Self-explanatory

Furthermore, if you set `ADDS_EXITS = True`, you'll need to define the method `static_exits()`. This function takes a single parameter, a list of IRSBs that would be executed in the run-up to your function, and asks you to return a list of all the exits that you know would be produced by your function in that case. The return value is expected to be a list of tuples of (address (int), jumpkind (str)). This is meant to be a quick, best-effort analysis, and you shouldn't try to do anything crazy or intensive to get your answer.

### 6.1.7 User Hooks

The process of writing and using a SimProcedure makes a lot of assumptions that you want to hook over a whole function. What if you don't? There's an alternate interface for hooking, a *user hook*, that lets you streamline the process of hooking sections of code.

```
>>> @project.hook(0x1234, length=5)
... def set_rax(state):
...     state.regs.rax = 1
```

This is a lot simpler! The idea is to use a single function instead of an entire SimProcedure subclass. No extraction of arguments is performed, no complex control flow happens.

Control flow is controlled by the length argument. After the function finishes executing in this example, the next step will start at 5 bytes after the hooked address. If the length argument is omitted or set to zero, execution will resume executing the binary code at exactly the hooked address, without re-triggering the hook. The `Ijk_NoHook` jumpkind allows this to happen.

If you want more control over control flow coming out of a user hook, you can return a list of successor states. Each successor will be expected to have `state.regs.ip`, `state.scratch.guard`, and `state.scratch.jumpkind` set. The IP is the target instruction pointer, the guard is a symbolic boolean representing a constraint to add to the state related to it being taken as opposed to the others, and the jumpkind is a VEX enum string, like `Ijk_Boring`, representing the nature of the branch.

The general rule is, if you want your SimProcedure to either be able to extract function arguments or cause a program return, write a full SimProcedure class. Otherwise, use a user hook.

### 6.1.8 Hooking Symbols

As you should recall from the *section on loading a binary*, dynamically linked programs have a list of symbols that they must import from the libraries they have listed as dependencies, and angr will make sure, rain or shine, that every import symbol gets resolved by *some* address, whether it's a real implementation of the function or just a dummy address hooked with a do-nothing stub. As a result, you can just use the `Project.hook_symbol` API to hook the address referred to by a symbol!

This means that you can replace library functions with your own code. For instance, to replace `rand()` with a function that always returns a consistent sequence of values:

```
>>> class NotVeryRand(SimProcedure):
...     def run(self, return_values=None):
...         rand_idx = self.state.globals.get('rand_idx', 0) % len(return_values)
...         out = return_values[rand_idx]
...         self.state.globals['rand_idx'] = rand_idx + 1
...         return out

>>> project.hook_symbol('rand', NotVeryRand(return_values=[413, 612, 1025, 1111]))
```

Now, whenever the program tries to call `rand()`, it'll return the integers from the `return_values` array in a loop.

## 6.2 State Plugins

If you want to store some data on a state and have that information propagated from successor to successor, the easiest way to do this is with `state.globals`. However, this can become obnoxious with large amounts of interesting data, doesn't work at all for merging states, and isn't very object-oriented.

The solution to these problems is to write a *State Plugin* - an appendix to the state that holds data and implements an interface for dealing with the lifecycle of a state.

### 6.2.1 My First Plugin

Let's get started! All state plugins are implemented as subclasses of `SimStatePlugin`. Once you've read this document, you can use the API reference for this class *`angr.state_plugins.plugin.SimStatePlugin`* to quickly review the semantics of all the interfaces you should implement.

The most important method you need to implement is `copy`: it should be annotated with the `memo` staticmethod and take a dict called the "memo"—these'll be important later—and returns a copy of the plugin. Short of that, you can do whatever you want. Just make sure to call the superclass initializer!

```
>>> import angr
>>> class MyFirstPlugin(angr.SimStatePlugin):
...     def __init__(self, foo):
...         super(MyFirstPlugin, self).__init__()
...         self.foo = foo
...
...     @angr.SimStatePlugin.memo
...     def copy(self, memo):
...         return MyFirstPlugin(self.foo)

>>> project = angr.load_shellcode(b'\x90', 'AMD64')
>>> state = angr.SimState(project=project)
>>> state.register_plugin('my_plugin', MyFirstPlugin('bar'))
>>> assert state.my_plugin.foo == 'bar'
```

(continues on next page)

(continued from previous page)

```
>>> state2 = state.copy()
>>> state.my_plugin.foo = 'baz'
>>> state3 = state.copy()
>>> assert state2.my_plugin.foo == 'bar'
>>> assert state3.my_plugin.foo == 'baz'
```

It works! Note that plugins automatically become available as attributes on the state. `state.get_plugin(name)` is also available as a more programmatic interface.

### 6.2.2 Where's the state?

State plugins have access to the state, right? So why isn't it part of the initializer? It turns out, there are a plethora of issues related to initialization order and dependency issues, so to simplify things as much as possible, the state is not part of the initializer but is rather set onto the state in a separate phase, by using the `set_state` method. You can override this state if you need to do things like propagate the state to subcomponents or extract architectural information.

```
>>> def set_state(self, state):
...     super(SimStatePlugin, self).set_state(state)
...     self.symbolic_word = claripy.BVS('my_variable', self.state.arch.bits)
```

Note the `self.state`! That's what the super `set_state` sets up.

However, there's no guarantee on what order the states will be set onto the plugins in, so if you need to interact with *other plugins* for initialization, you need to override the `init_state` method.

Once again, there's no guarantee on what order these will be called in, so the rule is to make sure you set yourself up good enough during `set_state` so that if someone else tries to interact with you, no type errors will happen. Here's an example of a good use of `init_state`, to map a memory region in the state. The use of an instance variable (presumably copied as part of `copy()`) ensures this only happens the first time the plugin is added to a state.

```
>>> def init_state(self):
...     if self.region is None:
...         self.region = self.state.memory.map_region(SOMEWHERE, 0x1000, 7)
```

#### Note: weak references

`self.state` is not the state itself, but rather a weak proxy to the state. You can still use this object as a normal state, but attempts to store it persistently will not work.

### 6.2.3 Merging

The other element besides copying in the state lifecycle is merging. As input you get the plugins to merge and a list of "merge conditions" - symbolic booleans that are the "guard conditions" describing when the values from each state should actually apply.

The important properties of the merge conditions are:

- They are mutually exclusive and span an entire domain - exactly one may be satisfied at once, and there will be additional constraints to ensure that at least one must be satisfied.
- `len(merge_conditions)` == len(others) + 1, since `self` counts too.
- `zip(merge_conditions, [self] + others)` will correctly pair merge conditions with plugins.

During the merge function, you should *mutate* `self` to become the merged version of itself and all the others, with respect to the merge conditions. This involves using the if-then-else structure that claripy provides. Here is an example

of constructing this merged structure by merging a bitvector instance variable called `myvar`, producing a binary tree of if-then-else expressions searching for the correct condition:

```
for other_plugin, condition in zip(others, merge_conditions[1:]): # chop off self's␣
↪condition
    self.myvar = claripy.If(condition, other_plugin.myvar, self.myvar)
```

This is such a common construction that we provide a utility to perform it automatically: `claripy.ite_cases`. The following code snippet is identical to the previous one:

```
self.myvar = claripy.ite_cases(zip(merge_conditions[1:], [o.myvar for o in others]),␣
↪self.myvar)
```

Keep in mind that like the rest of the top-level claripy functions, `ite_cases` and `If` are also available from `state.solver`, and these versions will perform SimActionObject unwrapping if applicable.

#### Common Ancestor

The full prototype of the `merge` interface is `def merge(self, others, merge_conditions, common_ancestor=None)`. `others` and `merge_conditions` have been discussed in depth already.

The common ancestor is the instance of the plugin from the most recent common ancestor of the states being merged. It may not be available for all merges, in which case it will be None. There are no rules for how exactly you should use this to improve the quality of your merges, but you may find it useful in more complex setups.

### 6.2.4 Serialization

In order to support serialization of states which contain your plugin, you should implement the `__getstate__`/`__setstate__` magic method pair. Keep in mind the following guidelines:

- Your serialization result should *not* include the state.
- After deserialization, `set_state()` will be called again.

This means that plugins are "detached" from the state and serialized in an isolated environment, and then reattached to the state on deserialization.

### 6.2.5 Plugins all the way down

You may have components within your state plugins which are large and complicated and start breaking object-orientation in order to make copy/merge work well with the state lifecycle. You're in luck! Things can be state plugins even if they aren't directly attached to a state. A great example of this is `SimFile`, which is a state plugin but is stored in the filesystem plugin, and is never used with `SimState.register_plugin`. When you're doing this, there are a handful of rules to remember which will keep your plugins safe and happy:

- Annotate your copy function with `@SimStatePlugin.memo`.
- In order to prevent *divergence* while copying multiple references to the same plugin, make sure you're passing the memo (the argument to copy) to the `.copy` of any subplugins. This with the previous point will preserve object identity.
- In order to prevent *duplicate merging* while merging multiple references to the same plugin, there should be a concept of the "owner" of each instance, and only the owner should run the merge routine.
- While passing arguments down into sub-plugins `merge()` routines, make sure you unwrap `others` and `common_ancestor` into the appropriate types. For example, if `PluginA` contains a `PluginB`, the former should do the following:

```
>>> def merge(self, others, merge_conditions, common_ancestor=None):
...     # ... merge self
...     self.plugin_b.merge([o.plugin_b for o in others], merge_conditions,
...         common_ancestor=None if common_ancestor is None else common_ancestor.plugin_
↪b)
```

### 6.2.6 Setting Defaults

To make it so that a plugin will automatically become available on a state when requested, without having to register it with the state first, you can register it as a *default*. The following code example will make it so that whenever you access `state.my_plugin`, a new instance of `MyPlugin` will be instantiated and registered with the state.

```
MyPlugin.register_default('my_plugin')
```

## 6.3 Extending the Environment Model

One of the biggest issues you may encounter while using angr to analyze programs is an incomplete model of the environment, or the APIs, surrounding your program. This usually takes the form of syscalls or dynamic library calls, or in rare cases, loader artifacts. angr provides a convenient interface to do most of these things!

Everything discussed here involves writing SimProcedures, so *make sure you know how to do that!*.

Note that this page should be treated as a narrative document, not a reference document, so you should read it at least once start to end.

### 6.3.1 Setup

You *probably* want to have a development install of angr, i.e. set up with the script in the angr-dev repository. It is remarkably easy to add new API models by just implementing them in certain folders of the angr repository. This is also desirable because any work you do in this field will almost always be useful to other people, and this makes it extremely easy to submit a pull request.

However, if you want to do your development out-of-tree, you want to work against a production version of angr, or you want to make customized versions of already-implemented API functions, there are ways to incorporate your extensions programmatically. Both these techniques, in-tree and out-of-tree, will be documented at each step.

### 6.3.2 Dynamic library functions - import dependencies

This is the easiest case, and the case that SimProcedures were originally designed for.

First, you need to write a SimProcedure representing the function. Then you need to let angr know about it.

#### Case 1, in-tree development: SimLibraries and catalogues

angr has a magical folder in its repository, angr/procedures. Within it are all the SimProcedure implementations that come bundled with angr as well as information about what libraries implement what functions.

Each folder in the `procedures` directory corresponds to some sort of *standard*, or a body that specifies the interface part of an API and its semantics. We call each folder a *catalog* of procedures. For example, we have `libc` which contains the functions defined by the C standard library, and a separate folder `posix` which contains the functions defined by the posix standard. There is some magic which automatically scrapes these folders in the `procedures` directory and organizes them into the `angr.SIM_PROCEDURES` dict. For example, `angr/procedures/libc/printf.py` contains both `class printf` and `class __printf_chk`, so there exists both `angr.SIM_PROCEDURES['libc']['printf']` and `angr.SIM_PROCEDURES['libc']['__printf_chk']`.

The purpose of this categorization is to enable easy sharing of procedures among different libraries. For example. libc.so.6 contains all the C standard library functions, but so does msvcrt.dll! These relationships are represented with objects called `SimLibraries` which represent an actual shared library file, its functions, and their metadata. Take a look at the API reference for *SimLibrary* along with the code for setting up glibc to learn how to use it.

SimLibraries are defined in a special folder in the procedures directory, `procedures/definitions`. Files in here should contain an *instance*, not a subclass, of `SimLibrary`. The same magic that scrapes up SimProcedures will also scrape up SimLibraries and put them in `angr.SIM_LIBRARIES`, keyed on each of their common names. For example, `angr/procedures/definitions/linux_loader.py` contains `lib = SimLibrary(); lib.set_library_names('ld.so', 'ld-linux.so', 'ld.so.2', 'ld-linux.so.2', 'ld-linux-x86_64.so.2')`, so you can access it via `angr.SIM_LIBRARIES['ld.so'][0]` or `angr.SIM_LIBRARIES['ld-linux.so'][0]` or any of the other names.

At load time, all the dynamic library dependencies are looked up in `SIM_LIBRARIES` and their procedures (or stubs!) are hooked into the project's address space to summarize any functions it can. The code for this process is found here.

**SO**, the bottom line is that you can just write your own SimProcedure and SimLibrary definitions, drop them into the directory structure, and they'll automatically be applied. If you're adding a procedure to an existing library, you can just drop it into the appropriate catalog and it'll be picked up by all the libraries using that catalog, since most libraries construct their list of function implementation by batch-adding entire catalogs.

#### Case 2, out-of-tree development, tight integration

If you'd like to implement your procedures outside the angr repository, you can do that. You effectively do this by just manually adding your procedures to the appropriate SimLibrary. Just call `angr.SIM_LIBRARIES[libname][0].add(name, proc_cls)` to do the registration.

Note that this will only work if you do this before the project is loaded with `angr.Project`. Note also that adding the procedure to `angr.SIM_PROCEDURES`, i.e. adding it directly to a catalog, will *not* work, since these catalogs are used to construct the SimLibraries only at import and are used by value, not by reference.

#### Case 3, out-of-tree development, loose integration

Finally, if you don't want to mess with SimLibraries at all, you can do things purely on the project level with *hook_symbol()*.

### 6.3.3 Syscalls

Unlike dynamic library methods, syscall procedures aren't incorporated into the project via hooks. Instead, whenever a syscall instruction is encountered, the basic block should end with a jumpkind of `Ijk_Sys`. This will cause the next step to be handled by the SimOS associated with the project, which will extract the syscall number from the state and query a specialized SimLibrary with that.

This deserves some explanation.

There is a subclass of SimLibrary called SimSyscallLibrary which is used for collecting all the functions that are part of an operating system's syscall interface. SimSyscallLibrary uses the same system for managing implementations and metadata as SimLibrary, but adds on top of it a system for managing syscall numbers for multiple ABIs (application binary interfaces, like an API but lower level). The best example for an implementation of a SimSyscallLibrary is the linux syscalls. It keeps its procedures in a normal SimProcedure catalog called `linux_kernel` and adds them to the library, then adds several syscall number mappings, including separate mappings for `mips-o32`, `mips-n32`, and `mips-n64`.

In order for syscalls to be supported in the first place, the project's SimOS must inherit from *SimUserland*, itself a SimOS subclass. This requires the class to call SimUserland's constructor with a super() call that includes the `syscall_library` keyword argument, specifying the specific SimSyscallLibrary that contains the appropriate procedures and mappings for the operating system. Additionally, the class's `configure_project` must perform a super()

call including the `abi_list` keyword argument, which contains the list of ABIs that are valid for the current architecture. If the ABI for the syscall can't be determined by just the syscall number, for example, that amd64 linux programs can use either `int 0x80` or `syscall` to invoke a syscall and these two ABIs use overlapping numbers, the SimOS cal override `syscall_abi()`, which takes a SimState and returns the name of the current syscall ABI. This is determined for int80/syscall by examining the most recent jumpkind, since libVEX will produce different syscall jumpkinds for the different instructions.

Calling conventions for syscalls are a little weird right now and they ought to be refactored. The current situation requires that `angr.SYSCALL_CC` be a map of maps `{arch_name: {os_name: cc_cls}}`, where `os_name` is the value of project.simos.name, and each of the calling convention classes must include an extra method called `syscall_number` which takes a state and return the current syscall number. Look at the bottom of calling_conventions.py to learn more about it. Not very object-oriented at all...

As a side note, each syscall is given a unique address in a special object in CLE called the "kernel object". Upon a syscall, the address for the specific syscall is set into the state's instruction pointer, so it will show up in the logs. These addresses are not hooked, they are just used to identify syscalls during analysis given only an address trace. The test for determining if an address corresponds to a syscall is `project.simos.is_syscall_addr(addr)` and the syscall corresponding to the address can be retrieved with `project.simos.syscall_from_addr(addr)`.

### Case 1, in-tree development

SimSyscallLibraries are stored in the same place as the normal SimLibraries, `angr/procedures/definitions`. These libraries don't have to specify any common name, but they can if they'd like to show up in `SIM_LIBRARIES` for easy access.

The same thing about adding procedures to existing catalogs of dynamic library functions also applies to syscalls - implementing a linux syscall is as easy as writing the SimProcedure and dropping the implementation into `angr/procedures/linux_kernel`. As long as the class name matches one of the names in the number-to-name mapping of the SimLibrary (all the linux syscall numbers are included with recent releases of angr), it will be used.

To add a new operating system entirely, you need to implement the SimOS as well, as a subclass of SimUserland. To integrate it into the tree, you should add it to the `simos` directory, but this is not a magic directory like `procedures`. Instead, you should add a line to `angr/simos/__init__.py` calling `register_simos()` with the OS name as it appears in `project.loader.main_object.os` and the SimOS class. Your class should do everything described above.

### Case 2, out-of-tree development, tight integration

You can add syscalls to a SimSyscallLibrary the same way you can add functions to a normal SimLibrary, by tweaking the entries in `angr.SIM_LIBRARIES`. If you're this for linux you want `angr.SIM_LIBRARIES['linux'][0].add(name, proc_cls)`.

You can register a SimOS with angr from out-of-tree as well - the same `register_simos` method is just sitting there waiting for you as `angr.simos.register_simos(name, simos_cls)`.

### Case 3, out-of-tree development, loose integration

The SimSyscallLibrary the SimOS uses is copied from the original during setup, so it is safe to mutate. You can directly fiddle with `project.simos.syscall_library` to manipulate an individual project's syscalls.

You can provide a SimOS class (not an instance) directly to the `Project` constructor via the `simos` keyword argument, so you can specify the SimOS for a project explicitly if you like.

### 6.3.4 SimData

What about when there is an import dependency on a data object? This is easily resolved when the given library is actually loaded into memory - the relocation can just be resolved as normal. However, when the library is not loaded (for example, `auto_load_libs=False`, or perhaps some dependency is simply missing), things get tricky. It is not possible to guess in most cases what the value should be, or even what its size should be, so if the guest program ever dereferences a pointer to such a symbol, emulation will go off the rails.

CLE will warn you when this might happen:

```
[22:26:58] [cle.backends.externs] |  WARNING: Symbol was allocated without a known size;␣
↪emulation will fail if it is used non-opaquely: _rtld_global
[22:26:58] [cle.backends.externs] |  WARNING: Symbol was allocated without a known size;␣
↪emulation will fail if it is used non-opaquely: __libc_enable_secure
[22:26:58] [cle.backends.externs] |  WARNING: Symbol was allocated without a known size;␣
↪emulation will fail if it is used non-opaquely: _rtld_global_ro
[22:26:58] [cle.backends.externs] |  WARNING: Symbol was allocated without a known size;␣
↪emulation will fail if it is used non-opaquely: _dl_argv
```

If you see this message and suspect it is causing issues (i.e. the program is actually introspecting the value of these symbols), you can resolve it by implementing and registering a SimData class, which is like a SimProcedure but for data. Simulated data. Very cool.

A SimData can effectively specify some data that must be used to provide an unresolved import symbol. It has a number of mechanisms to make this more useful, including the ability to specify relocations and subdependencies.

Look at the SimData `cle.backends.externs.simdata.SimData` class reference and the existing SimData subclasses for guidelines on how to do this.

## 6.4 Writing Analyses

An analysis can be created by subclassing the `angr.Analysis` class. In this section, we'll create a mock analysis to show off the various features. Let's start with something simple:

```
>>> import angr

>>> class MockAnalysis(angr.Analysis):
...     def __init__(self, option):
...         self.option = option

>>> angr.AnalysesHub.register_default('MockAnalysis', MockAnalysis) # register the class␣
↪with angr's global analysis list
```

This is a very simple analysis – it takes an option, and stores it. Of course, it's not useful, but this is just a demonstration.

Let's see how to run our new analysis:

```
>>> proj = angr.Project("/bin/true")
>>> mock = proj.analyses.MockAnalysis('this is my option')
>>> assert mock.option == 'this is my option'
```

### 6.4.1 Working with projects

Via some Python magic, your analysis will automatically have the project upon which you are running it under the `self.project` property. Use this to interact with your project and analyze it!

```
>>> class ProjectSummary(angr.Analysis):
...     def __init__(self):
...         self.result = 'This project is a %s binary with an entry point at %#x.' %␣
↪(self.project.arch.name, self.project.entry)

>>> angr.AnalysesHub.register_default('ProjectSummary', ProjectSummary)
>>> proj = angr.Project("/bin/true")

>>> summary = proj.analyses.ProjectSummary()
>>> print(summary.result)
This project is a AMD64 binary with an entry point at 0x401410.
```

### 6.4.2 Analysis Resilience

Sometimes, your (or our) code might suck and analyses might throw exceptions. We understand, and we also understand that oftentimes a partial result is better than nothing. This is specifically true when, for example, running an analysis on all of the functions in a program. Even if some of the functions fails, we still want to know the results of the functions that do not.

To facilitate this, the `Analysis` base class provides a resilience context manager under `self._resilience`. Here's an example:

```
>>> class ComplexFunctionAnalysis(angr.Analysis):
...     def __init__(self):
...         self._cfg = self.project.analyses.CFG()
...         self.results = { }
...         for addr, func in self._cfg.function_manager.functions.items():
...             with self._resilience():
...                 if addr % 2 == 0:
...                     raise ValueError("can't handle functions at even addresses")
...                 else:
...                     self.results[addr] = "GOOD"
```

The context manager catches any exceptions thrown and logs them (as a tuple of the exception type, message, and traceback) to `self.errors`. These are also saved and loaded when the analysis is saved and loaded (although the traceback is discarded, as it is not picklable).

You can tune the effects of the resilience with two optional keyword parameters to `self._resilience()`.

The first is `name`, which affects where the error is logged. By default, errors are placed in `self.errors`, but if `name` is provided, then instead the error is logged to `self.named_errors`, which is a dict mapping `name` to a list of all the errors that were caught under that name. This allows you to easily tell where thrown without examining its traceback.

The second argument is `exception`, which should be the type of the exception that `resilience` should catch. This defaults to `Exception`, which handles (and logs) almost anything that could go wrong. You can also pass a tuple of exception types to this option, in which case all of them will be caught.

Using `resilience` has a few advantages:

1. Your exceptions are gracefully logged and easily accessible afterwards. This is really nice for writing testcases.
2. When creating your analysis, the user can pass `fail_fast=True`, which transparently disable the resilience, which is really nice for manual testing.

3. It's prettier than having `try` `except` everywhere.

Have fun with analyses! Once you master the rest of angr, you can use analyses to understand anything computable!

CHAPTER

SEVEN

# ANGR EXAMPLES

To help you get started with angr, we've created several examples. We've tried to organize them into major categories, and briefly summarize that each example will expose you to. Enjoy!

If you want a high-level cheatsheet of the "techniques" used in the examples, see the angr strategies cheatsheet by Florent Bordignon.

To jump to a specific category:

- *Introduction* - examples showing off the very basics of angr's functionality
- *Reversing* - examples showing angr being used in reverse engineering tasks
- *Vulnerability Discovery* - examples of angr being used to search for vulnerabilities
- *Exploitation* - examples of angr being used as an exploitation assistance tool

## 7.1 Introduction

These are some introductory examples to give an idea of how to use angr's API.

### 7.1.1 Fauxware

This is a basic script that explains how to use angr to symbolically execute a program and produce concrete input satisfying certain conditions.

Binary, source, and script are found here.

## 7.2 Reversing

These are examples that use angr to solve reverse engineering challenges. There are a lot of these. We've chosen the most unique ones, and relegated the rest to the CTF Challenges section below.

### 7.2.1 Beginner reversing example: little_engine

```
Script author: Michael Reeves (github: @mastermjr)
Script runtime: 3 min 26 seconds (206 seconds)
Concepts presented:
stdin constraining, concrete optimization with Unicorn
```

This challenge is similar to the csaw challenge below, however the reversing is much more simple. The original code, solution, and writeup for the challenge can be found at the b01lers github here.

The angr solution script is here and the binary is here.

### 7.2.2 Whitehat CTF 2015 - Crypto 400

```
Script author: Yan Shoshitaishvili (github: @Zardus)
Script runtime: 30 seconds
Concepts presented: statically linked binary (manually hooking with function summaries),␣
↪commandline argument, partial solutions
```

We solved this crackme with angr's help. The resulting script will help you understand how angr can be used for crackme *assistance*, not a full-out solve. Since angr cannot solve the actual crypto part of the challenge, we use it just to reduce the keyspace, and brute-force the rest.

You can find this script here and the binary here.

### 7.2.3 CSAW CTF 2015 Quals - Reversing 500, "wyvern"

```
Script author: Audrey Dutcher (github: @rhelmot)
Script runtime: 15 mins
Concepts presented: stdin constraining, concrete optimization with Unicorn
```

angr can outright solve this challenge with very little assistance from the user. The script to do so is *here <https://github.com/angr/angr-examples/tree/master/examples/csaw_wyvern/solve.py>_* and the binary is here.

### 7.2.4 TUMCTF 2016 - zwiebel

```
Script author: Fish
Script runtime: 2 hours 31 minutes with pypy and Unicorn - expect much longer with␣
↪CPython only
Concepts presented: self-modifying code support, concrete optimization with Unicorn
```

This example is of a self-unpacking reversing challenge. This example shows how to enable Unicorn support and self-modification support in angr. Unicorn support is essential to solve this challenge within a reasonable amount of time - simulating the unpacking code symbolically is *very* slow. Thus, we execute it concretely in unicorn/qemu and only switch into symbolic execution when needed.

You may refer to other writeup about the internals of this binary. I didn't reverse too much since I was pretty confident that angr is able to solve it :-)

The long-term goal of optimizing angr is to execute this script within 10 minutes. Pretty ambitious :P

Here is the binary and the script.

### 7.2.5 FlareOn 2015 - Challenge 5

```
Script author: Adrian Tang (github: @tangabc)
Script runtime: 2 mins 10 secs
Concepts presented: Windows support
```

This is another reversing challenge from the FlareOn challenges.

"The challenge is designed to teach you about PCAP file parsing and traffic decryption by reverse engineering an executable used to generate it. This is a typical scenario in our malware analysis practice where we need to figure out precisely what the malware was doing on the network"

For this challenge, the author used angr to represent the desired encoded output as a series of constraints for the SAT solver to solve for the input.

For a detailed write-up please visit the author's post here and you can also find the solution from the FireEye here

### 7.2.6 0ctf quals 2016 - trace

```
Script author: WGH (wgh@bushwhackers.ru)
Script runtime: 1 min 50 secs (CPython 2.7.10), 1 min 12 secs (PyPy 4.0.1)
Concepts presented: guided symbolic tracing
```

In this challenge we're given a text file with trace of a program execution. The file has two columns, address and instruction executed. So we know all the instructions being executed, and which branches were taken. But the initial data is not known.

Reversing reveals that a buffer on the stack is initialized with known constant string first, then an unknown string is appended to it (the flag), and finally it's sorted with some variant of quicksort. And we need to find the flag somehow.

angr easily solves this problem. We only have to direct it to the right direction at every branch, and the solver finds the flag at a glance.

Files are here.

### 7.2.7 ASIS CTF Finals 2015 - license

```
Script author: Fish Wang (github: @ltfish)
Script runtime: 3.6 sec
Concepts presented: using the filesystem, manual symbolic summary execution
```

This is a crackme challenge that reads a license file. Rather than hooking the read operations of the flag file, we actually pass in a filesystem with the correct file created.

Here is the binary and the script.

### 7.2.8 DEFCON Quals 2017 - Crackme2000

```
Script author: Shellphish
Script runtime: varies, but on the order of seconds
Concepts presented: automated reverse engineering
```

DEFCON Quals had a whole category for automatic reversing in 2017. Our scripts are here.

## 7.3 Vulnerability Discovery

These are examples of angr being used to identify vulnerabilities in binaries.

### 7.3.1 Beginner vulnerability discovery example: strcpy_find

```
Script author: Kyle Ossinger (github: @k0ss)
Concepts presented: exploration to vulnerability, programmatic find condition
```

This is the first in a series of "tutorial scripts" I'll be making which use angr to find exploitable conditions in binaries. The first example is a very simple program. The script finds a path from the main entry point to `strcpy`, but **only** when we control the source buffer of the `strcpy` operation. To hit the right path, angr has to solve for a password argument, but angr solved this in less than 2 seconds on my machine using the standard Python interpreter. The script might look large, but that's only because I've heavily commented it to be more helpful to beginners. The challenge binary is here and the script is here.

### 7.3.2 CGC crash identification

```
Script author: Antonio Bianchi, Jacopo Corbetta
Concepts presented: exploration to vulnerability
```

This is a very easy binary containing a stack buffer overflow and an easter egg. CADET_00001 is one of the challenge released by DARPA for the Cyber Grand Challenge: link The binary can run in the DECREE VM: link A copy of the original challenge and the angr solution is provided here CADET_00001.adapted (by Jacopo Corbetta) is the same program, modified to be runnable in an Intel x86 Linux machine.

### 7.3.3 Grub "back to 28" bug

```
Script author: Audrey Dutcher (github: @rhelmot)
Concepts presented: unusual target (custom function hooking required), use of␣
↪exploration techniques to categorize and prune the program's state space
```

This is the demonstration presented at 32c3. The script uses angr to discover the input to crash grub's password entry prompt.

script - vulnerable module

## 7.4 Exploitation

These are examples of angr's use as an exploitation assistance engine.

### 7.4.1 Insomnihack Simple AEG

```
Script author: Nick Stephens (github: @NickStephens)
Concepts presented: automatic exploit generation, global symbolic data tracking
```

Demonstration for Insomni'hack 2016. The script is a very simple implementation of AEG.

script

### 7.4.2 SecuInside 2016 Quals - mbrainfuzz - symbolic exploration for exploitability conditions

```
Script author: nsr (nsr@tasteless.eu)
Script runtime: ~15 seconds per binary
Concepts presented: symbolic exploration guided by static analysis, using the CFG
```

Originally, a binary was given to the ctf-player by the challenge-service, and an exploit had to be crafted automatically. Four sample binaries, obtained during the ctf, are included in the example. All binaries follow the same format; the command-line argument is validated in a bunch of functions, and when every check succeeds, a memcpy() resulting into a stack-based buffer overflow is executed. angr is used to find the way through the binary to the memcpy() and to generate valid inputs to every checking function individually.

The sample binaries and the script are located here and additional information be found at the author's Write-Up.

### 7.4.3 SECCON 2016 Quals - ropsynth

```
Script author: Yan Shoshitaishvili (github @zardus) and Nilo Redini
Script runtime: 2 minutes
```

(continues on next page)

(continued from previous page)

```
Concepts presented: automatic ROP chain generation, binary modification, reasoning over␣
↪constraints, reasoning over action history
```

This challenge required the automatic generation of ropchains, with the twist that every ropchain was succeeded by an input check that, if not passed, would terminate the application. We used symbolic execution to recover those checks, removed the checks from the binary, used angrop to build the ropchains, and instrumented them with the inputs to pass the checks.

The various challenge files are located here, with the actual solve script here.

CHAPTER

EIGHT

# FREQUENTLY ASKED QUESTIONS

This is a collection of commonly-asked "how do I do X?" questions and other general questions about angr, for those too lazy to read this whole document.

If your question is of the form "how do I fix X issue after installing", see also the Troubleshooting section of the *install instructions*.

## 8.1 Why is it named angr?

The core of angr's analysis is on VEX IR, and when something is vexing, it makes you angry.

## 8.2 How should "angr" be stylized?

All lowercase, even at the beginning of sentences. It's an anti-proper noun.

## 8.3 Why isn't symbolic execution doing the thing I want?

The universal debugging technique for symbolic execution is as follows:

- Check your simulation manager for errored states. `print(simgr)` is a good place to start, and if you see anything to do with "errored", go for `print(simgr.errored)`.
- If you have any errored states and it's not immediately obvious what you did wrong, you can get a pdb shell at the crash site by going `simgr.errored[n].debug()`.
- If no state has reached an address you care about, you should check the path each state has gone down: `import pprint; pprint.pprint(state.history.descriptions.hardcopy)`. This will show you a high-level summary of what the symbolic execution engine did at each step along the state's history. You will be able to see from this a basic block trace and also a list of executed simprocedures. If you're using unicorn engine, you can check `state.history.bbl_addrs.hardcopy` to see what blocks were executed in each invocation of unicorn.
- If a state is going down the wrong path, you can check what constraints caused it to go that way: `print(state.solver.constraints)`. If a state has just gone past a branch, you can check the most recent branch condition with `state.history.events[-1]`.

## 8.4 How can I get diagnostic information about what angr is doing?

angr uses the standard `logging` module for logging, with every package and submodule creating a new logger.

The simplest way to get debug output is the following:

```
import logging
logging.getLogger('angr').setLevel('DEBUG')
```

You may want to use `INFO` or whatever else instead. By default, angr will enable logging at the `WARNING` level.

Each angr module has its own logger string, usually all the Python modules above it in the hierarchy, plus itself, joined with dots. For example, `angr.analyses.cfg`. Because of the way the Python logging module works, you can set the verbosity for all submodules in a module by setting a verbosity level for the parent module. For example, `logging.getLogger('angr.analyses').setLevel('INFO')` will make the CFG, as well as all other analyses, log at the INFO level.

## 8.5 Why is angr so slow?

It's complicated! *Optimization considerations*

## 8.6 How do I find bugs using angr?

It's complicated! The easiest way to do this is to define a "bug condition", for example, "the instruction pointer has become a symbolic variable", and run symbolic exploration until you find a state matching that condition, then dump the input as a testcase. However, you will quickly run into the state explosion problem. How you address this is up to you. Your solution may be as simple as adding an `avoid` condition or as complicated as implementing CMU's MAYHEM system as an Exploration Technique.

## 8.7 Why did you choose VEX instead of another IR (such as LLVM, REIL, BAP, etc)?

We had two design goals in angr that influenced this choice:

1. angr needed to be able to analyze binaries from multiple architectures. This mandated the use of an IR to preserve our sanity, and required the IR to support many architectures.
2. We wanted to implement a binary analysis engine, not a binary lifter. Many projects start and end with the implementation of a lifter, which is a time consuming process. We needed to take something that existed and already supported the lifting of multiple architectures.

Searching around the internet, the major choices were:

- LLVM is an obvious first candidate, but lifting binary code to LLVM cleanly is a pain. The two solutions are either lifting to LLVM through QEMU, which is hackish (and the only implementation of it seems very tightly integrated into S2E), or McSema, which only supported x86 at the time but has since gone through a rewrite and gotten support for x86-64 and aarch64.
- TCG is QEMU's IR, but extracting it seems very daunting as well and documentation is very scarce.
- REIL seems promising, but there is no standard reference implementation that supports all the architectures that we wanted. It seems like a nice academic work, but to use it, we would have to implement our own lifters, which we wanted to avoid.
- BAP was another possibility. When we started work on angr, BAP only supported lifting x86 code, and up-to-date versions of BAP were only available to academic collaborators of the BAP authors. These were two deal-breakers. BAP has since become open, but it still only supports x86_64, x86, and ARM.
- VEX was the only choice that offered an open library and support for many architectures. As a bonus, it is very well documented and designed specifically for program analysis, making it very easy to use in angr.

While angr uses VEX now, there's no fundamental reason that multiple IRs cannot be used. There are two parts of angr, outside of the `angr.engines.vex` package, that are VEX-specific:

- the jump labels (i.e., the `Ijk_Ret` for returns, `Ijk_Call` for calls, and so forth) are VEX enums.
- VEX treats registers as a memory space, and so does angr. While we provide accesses to `state.regs.rax` and friends, on the backend, this does `state.registers.load(8, 8)`, where the first `8` is a VEX-defined offset for `rax` to the register file.

To support multiple IRs, we'll either want to abstract these things or translate their labels to VEX analogues.

## 8.8 Why are some ARM addresses off-by-one?

In order to encode THUMB-ness of an ARM code address, we set the lowest bit to one. This convention comes from LibVEX, and is not entirely our choice! If you see an odd ARM address, that just means the code at `address - 1` is in THUMB mode.

## 8.9 How do I serialize angr objects?

Pickle will work. However, Python will default to using an extremely old pickle protocol that does not support more complex Python data structures, so you must specify a more advanced data stream format. The easiest way to do this is `pickle.dumps(obj, -1)`.

## 8.10 What does `UnsupportedIROpError("floating point support disabled")` mean?

This might crop up if you're using a CGC analysis such as driller or rex. Floating point support in angr has been disabled in the CGC analyses for a tight-knit nebula of reasons:

- Libvex's representation of floating point numbers is imprecise - it converts the 80-bit extended precision format used by the x87 for computation to 64-bit doubles, making it impossible to get precise results
- There is very limited implementation support in angr for the actual primitive operations themselves as reported by libvex, so you will often get a less friendly "unsupported operation" error if you go too much further
- For what operations are implemented, the basic optimizations that allow tractability during symbolic computation (AST deduplication, operation collapsing) are not implemented for floating point ops, leading to gigantic ASTs
- There are memory corruption bugs in z3 that get triggered frighteningly easily when you're using huge workloads of mixed floating point and bitvector ops. We haven't been able to get a testcase that doesn't involve "just run angr" for the z3 guys to investigate.

Instead of trying to cope with all of these, we have simply disabled floating point support in the symbolic execution engine. To allow for execution in the presence of floating point ops, we have enabled an exploration technique called the *https://github.com/angr/angr/blob/master/angr/exploration_techniques/oppologist.py <oppologist>* that is supposed to catch these issues, concretize their inputs, and run the problematic instructions through qemu via unicorn engine, allowing execution to continue. The intuition is that the specific values of floating point operations don't typically affect the exploitation process.

If you're seeing this error and it's terminating the analysis, it's probably because you don't have unicorn installed or configured correctly. If you're seeing this issue just in a log somewhere, it's just the oppologist kicking in and you have nothing to worry about.

## 8.11 Why is angr's CFG different from IDA's?

Two main reasons:

- IDA does not split basic blocks at function calls. angr will, because they are a form of control flow and basic blocks end at control flow instructions. You generally do not need the supergraph for performing automated analyses.
- IDA will split basic blocks if another block jumps into the middle of it. This is called basic block normalization, and angr does not do it by default since it is unnecessary for most static analyses. You may enable it by passing `normalize=True` to the CFG analysis.

## 8.12 Why do I get incorrect register values when reading from a state during a SimInspect breakpoint?

libVEX will eliminate duplicate register writes within a single basic block when optimizations are enabled. Turn off IR optimization to make everything look right at all times.

In the case of the instruction pointer, libVEX will frequently omit mid-block writes even when optimizations are disabled. In this case, you should use `state.scratch.ins_addr` to get the current instruction pointer.

CHAPTER

NINE

# APPENDIX

## 9.1 Cheatsheet

The following cheatsheet aims to give an overview of various things you can do with angr and act as a quick reference to check the syntax for something without having to dig through the deeper docs.

### 9.1.1 General getting started

Some useful imports

```
import angr #the main framework
import claripy #the solver engine
```

Loading the binary

```
proj = angr.Project("/path/to/binary", auto_load_libs=False) # auto_load_libs False for␣
↪improved performance
```

### 9.1.2 States

Create a SimState object

```
state = proj.factory.entry_state()
```

### 9.1.3 Simulation Managers

Generate a simulation manager object

```
simgr = proj.factory.simulation_manager(state)
```

### 9.1.4 Exploring and analysing states

Choosing a different Exploring strategy

```
simgr.use_technique(angr.exploration_techniques.DFS())
```

Symbolically execute until we find a state satisfying our `find=` and `avoid=` parameters

```
avoid_addr = [0x400c06, 0x400bc7]
find_addr = 0x400c10d
simgr.explore(find=find_addr, avoid=avoid_addr)
```

```
found = simgr.found[0] # A state that reached the find condition from explore
found.solver.eval(sym_arg, cast_to=bytes) # Return a concrete string value for the sym␣
↪arg to reach this state
```

Symbolically execute until lambda expression is `True`

```
simgr.step(until=lambda sm: sm.active[0].addr >= first_jmp)
```

This is especially useful with the ability to access the current STDOUT or STDERR (1 here is the File Descriptor for STDOUT)

```
simgr.explore(find=lambda s: "correct" in s.posix.dumps(1))
```

Memory Management on big searches (Auto Drop Stashes):

```
simgr.explore(find=find_addr, avoid=avoid_addr, step_func=lambda lsm: lsm.drop(stash=
↪'avoid'))
```

#### Manually Exploring

```
simgr.step(step_func=step_func, until=lambda lsm: len(sm.found) > 0)

def step_func(lsm):
    lsm.stash(filter_func=lambda state: state.addr == 0x400c06, from_stash='active', to_
↪stash='avoid')
    lsm.stash(filter_func=lambda state: state.addr == 0x400bc7, from_stash='active', to_
↪stash='avoid')
    lsm.stash(filter_func=lambda state: state.addr == 0x400c10, from_stash='active', to_
↪stash='found')
    return lsm
```

Enable Logging output from Simulation Manager:

```
import logging
logging.getLogger('angr.sim_manager').setLevel(logging.DEBUG)
```

#### Stashes

Move Stash:

```
simgr.stash(from_stash="found", to_stash="active")
```

Drop Stashes:

```
simgr.drop(stash="avoid")
```

### 9.1.5 Constraint Solver (claripy)

Create symbolic object

```
sym_arg_size = 15 #Length in Bytes because we will multiply with 8 later
sym_arg = claripy.BVS('sym_arg', 8*sym_arg_size)
```

Restrict sym_arg to typical char range

```
for byte in sym_arg.chop(8):
    initial_state.add_constraints(byte >= '\x20') # ' '
    initial_state.add_constraints(byte <= '\x7e') # '~'
```

Create a state with a symbolic argument

```
argv = [proj.filename]
argv.append(sym_arg)
state = proj.factory.entry_state(args=argv)
```

Use argument for solving:

```
sym_arg = angr.claripy.BVS("sym_arg", flag_size * 8)
argv = [proj.filename]
argv.append(sym_arg)
initial_state = proj.factory.full_init_state(args=argv, add_options=angr.options.unicorn,
↪ remove_options={angr.options.LAZY_SOLVES})
```

### 9.1.6 FFI and Hooking

Calling a function from ipython

```
f = proj.factory.callable(address)
f(10)
x=claripy.BVS('x', 64)
f(x) #TODO: Find out how to make that result readable
```

If what you are interested in is not directly returned because for example the function returns the pointer to a buffer you can access the state after the function returns with

```
>>> f.result_state
<SimState @ 0x1000550>
```

Hooking

There are already predefined hooks for libc functions (useful for statically compiled libraries)

```
proj = angr.Project('/path/to/binary', use_sim_procedures=True)
proj.hook(addr, angr.SIM_PROCEDURES['libc']['atoi']())
```

Hooking with Simprocedure:

```
class fixpid(angr.SimProcedure):
    def run(self):
            return 0x30

proj.hook(0x4008cd, fixpid())
```

### 9.1.7 Other useful tricks

Drop into an ipython if a ctr+c is received (useful for debugging scripts that are running forever)

```
import signal
def killmyself():
```

(continues on next page)

(continued from previous page)

```
    os.system('kill %d' % os.getpid())
def sigint_handler(signum, frame):
    print 'Stopping Execution for Debug. If you want to kill the program issue:␣
↪killmyself()'
    if not "IPython" in sys.modules:
        import IPython
        IPython.embed()

signal.signal(signal.SIGINT, sigint_handler)
```

Get the calltrace of a state to find out where we got stuck

```
state = simgr.active[0]
print state.callstack
```

Get a basic block

```
block = proj.factory.block(address)
block.capstone.pp() # Capstone object has pretty print and other data about the␣
↪disassembly
block.vex.pp()      # Print vex representation
```

### 9.1.8 State manipulation

Write to state:

```
aaaa = claripy.BVV(0x41414141, 32) # 32 = Bits
state.memory.store(0x6021f2, aaaa)
```

Read Pointer to Pointer from Frame:

```
poi1 = new_state.solver.eval(new_state.regs.rbp)-0x10
poi1 = new_state.mem[poi1].long.concrete
poi1 += 0x8
ptr1 = new_state.mem[poi1].long.concrete
```

Read from State:

```
key = []
for i in range(38):
    key.append(extractkey.mem[0x602140 + i*4].int.concrete)
```

Alternatively, the below expression is equivalent

```
key = extractkey.mem[0x602140].int.array(38).concrete
```

### 9.1.9 Debugging angr

Set Breakpoint at every Memory read/write:

```
new_state.inspect.b('mem_read', when=angr.BP_AFTER, action=debug_funcRead)
def debug_funcRead(state):
```

(continues on next page)

(continued from previous page)

```
    print 'Read', state.inspect.attrs.mem_read_expr, 'from', state.inspect.attrs.mem_
↪read_address
```

Set Breakpoint at specific Memory location:

```
new_state.inspect.b('mem_write', mem_write_address=0x6021f1, when=angr.BP_AFTER,␣
↪action=debug_funcWrite)
```

## 9.2 List of Claripy Operations

### 9.2.1 Arithmetic and Logic

| Name | Description | Example |
|---|---|---|
| LShR | Logically shifts an expression to the right. (the default shifts are arithmetic) | `x.LShR(10)` |
| RotateLeft | Rotates an expression left | `x.RotateLeft(8)` |
| RotateRight | Rotates an expression right | `x.RotateRight(8)` |
| And | Logical And (on boolean expressions) | `claripy.And(x == y, x > 0)` |
| Or | Logical Or (on boolean expressions) | `claripy.Or(x == y, y < 10)` |
| Not | Logical Not (on a boolean expression) | `claripy.Not(x == y)` is the same as `x != y` |
| If | An If-then-else | Choose the maximum of two expressions: `claripy.If(x > y, x, y)` |
| ULE | Unsigned less than or equal to | Check if x is less than or equal to y: `x.ULE(y)` |
| ULT | Unsigned less than | Check if x is less than y: `x.ULT(y)` |
| UGE | Unsigned greater than or equal to | Check if x is greater than or equal to y: `x.UGE(y)` |
| UGT | Unsigned greater than | Check if x is greater than y: `x.UGT(y)` |
| SLE | Signed less than or equal to | Check if x is less than or equal to y: `x.SLE(y)` |
| SLT | Signed less than | Check if x is less than y: `x.SLT(y)` |
| SGE | Signed greater than or equal to | Check if x is greater than or equal to y: `x.SGE(y)` |
| SGT | Signed greater than | Check if x is greater than y: `x.SGT(y)` |

**Todo**

Add the floating point ops

### 9.2.2 Bitvector Manipulation

| Name | Description | Example |
|---|---|---|
| SignExt | Pad a bitvector on the left with `n` sign bits | `x.sign_extend(n)` |
| ZeroExt | Pad a bitvector on the left with `n` zero bits | `x.zero_extend(n)` |
| Extract | Extracts the given bits (zero-indexed from the *right*, inclusive) from an expression. | Extract the least significant byte of x: `x[7:0]` |
| Concat | Concatenates any number of expressions together into a new expression. | `x.concat(y, ...)` |

### 9.2.3 Extra Functionality

There's a bunch of prepackaged behavior that you *could* implement by analyzing the ASTs and composing sets of operations, but here's an easier way to do it:

- You can chop a bitvector into a list of chunks of `n` bits with `val.chop(n)`
- You can endian-reverse a bitvector with `x.reversed`
- You can get the width of a bitvector in bits with `val.length`
- You can test if an AST has any symbolic components with `val.symbolic`
- You can get a set of the names of all the symbolic variables implicated in the construction of an AST with `val.variables`

## 9.3 List of State Options

### 9.3.1 State Modes

These may be enabled by passing `mode=xxx` to a state constructor.

| Mode name | Description |
|---|---|
| `symbolic` | The default mode. Useful for most emulation and analysis tasks. |
| `symbolic_appr` | Symbolic mode, but enables approximations for constraint solving. |
| `static` | A preset useful for static analysis. The memory model becomes an abstract region-mapping system, "fake return" successors skipping calls are added, and more. |
| `fastpath` | A preset for extremely lightweight static analysis. Executing will skip all intensive processing to give a quick view of the behavior of code. |
| `tracing` | A preset for attempting to execute concretely through a program with a given input. Enables unicorn, enables resilience options, and will attempt to emulate access violations correctly. |

### 9.3.2 Option Sets

These are sets of options, found as `angr.options.xxx`.

| Set name | Description |
|---|---|
| `common_o` | Options necessary for basic execution |
| `symbolic` | Options necessary for basic symbolic execution |
| `resilien` | Options that harden angr's emulation against unsupported operations, attempting to carry on by treating the result as an unconstrained symbolic value and logging the occasion to `state.history.events`. |
| `refs` | Options that cause angr to keep a log of all the memory, register, and temporary references complete with dependency information in `history.actions`. This option consumes a lot of memory, so be careful! |
| `approxim` | Options that enable approximations of constraint solves via value-set analysis instead of calling into z3 |
| `simplifi` | Options that cause data to be run through z3's simplifiers before it reaches memory or register storage |
| `unicorn` | Options that enable the unicorn engine for executing on concrete data |

### 9.3.3 Options

These are individual option objects, found as `angr.options.XXX`.

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| ABSTRACT_MEMORY | Use SimAbstractMemory to model memory as discrete regions | | static | |
| ABSTRACT_SOLVER | Allow splitting constraint sets during simplification | | static | |
| ACTION_DEPS | Track dependencies in SimActions | | | |
| APPROXIMATE_GUARD | Use VSA when evaluating guard conditions | | | |
| APPROXIMATE_MEMOR | Use VSA when evaluating memory indices | approximation | symbolic_approxim | |
| APPROXIMATE_MEMOR | Use VSA when evaluating memory load/store sizes | approximation | symbolic_approxim | |
| APPROXIMATE_SATIS | Use VSA when evaluating state satisfiability | approximation | symbolic_approxim | |
| AST_DEPS | Enables dependency tracking for all claripy ASTs | | | During execution |
| AUTO_REFS | An internal option used to track dependencies in SimProcedures | | | During execution |
| AVOID_MULTIVALUED | Return a symbolic value without touching memory for any read that has a symbolic address | | fastpath | |
| AVOID_MULTIVALUED | Do not perform any write that has a symbolic address | | fastpath | |
| BEST_EFFORT_MEMOR | Handle huge writes of symbolic size by pretending they are actually smaller | | static, fastpath | |
| BREAK_SIRSB_END | Debug: trigger a breakpoint at the end of each block | | | |
| BREAK_SIRSB_START | Debug: trigger a breakpoint at the start of each block | | | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| BREAK_SIRSTMT_END | Debug: trigger a breakpoint at the end of each IR statement | | | |
| BREAK_SIRSTMT_STA | Debug: trigger a breakpoint at the start of each IR statement | | | |
| BYPASS_ERRORED_IR | Treat clean helpers that fail with errors as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_ERRORED_IR | Treat operations that fail with errors as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_UNSUPPORTE | Treat unsupported clean helpers as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_UNSUPPORTE | Treat unsupported dirty helpers as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_UNSUPPORTE | Treat unsupported IR expressions as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_UNSUPPORTE | Treat unsupported operations as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_UNSUPPORTE | Treat unsupported IR statements as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_UNSUPPORTE | Treat unsupported syscalls as returning unconstrained symbolic values | resilience | fastpath, tracing | |
| BYPASS_VERITESTIN | Discard emulation errors during veritesting | resilience | fastpath, tracing | |
| CACHELESS_SOLVER | enable SolverCacheless | | | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| CALLLESS | Emulate call instructions as an unconstraining of the return value register | | | |
| CGC_ENFORCE_FD | CGC: make sure all reads and writes go to stdin and stdout, respectively | | | |
| CGC_NON_BLOCKING_ | CGC: always report "data available" in fdwait | | | |
| CGC_NO_SYMBOLIC_R | CGC: always read the maximum amount of data requested in the receive syscall | | | |
| COMPOSITE_SOLVER | Enable SolverComposite for independent constraint set optimization | symbolic | all except static | |
| CONCRETIZE | Concretize all symbolic expressions encountered during emulation | | | |
| CONCRETIZE_SYMBOL | Concreteize the sizes of file reads | | | |
| CONCRETIZE_SYMBOL | Concretize the sizes of symbolic writes to memory | | | |
| CONSERVATIVE_READ | Do not use SimConcretizationStrategyAny for reads; in case of read address concretization failures, return an unconstrained symbolic value | | | |
| CONSERVATIVE_WRIT | Do not use SimConcretizationStrategyAny for writes; in case of write address concretization failures, treat the store as a no-op | | | |
| CONSTRAINT_TRACKI | Set track=True for making claripy Solvers; enable use of unsat_core | | | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| COW_STATES | Copy states instead of mutating the initial state directly | common_options | all | |
| DOWNSIZE_Z3 | Downsize the claripy solver whenever possible to save memory | | | |
| DO_CCALLS | Perform IR clean calls | symbolic | all except fastpath | |
| DO_GETS | Perform IR register reads | common_options | all | |
| DO_LOADS | Perform IR memory loads | common_options | all | |
| DO_OPS | Perform IR computation operations | common_options | all | |
| DO_PUTS | Perform IR register writes | common_options | all | |
| DO_RET_EMULATION | For each Ijk_Call successor, add a corresponding Ijk_FakeRet successor | | static, fastpath | |
| DO_STORES | Perform IR memory stores | common_options | all | |
| EFFICIENT_STATE_M | Keep in memory any state that might be a common ancestor in a merge | | | Veritesting |
| ENABLE_NX | When in conjunction with STRICT_PAGE_ACCES raise a SimSegfaultException on executing non-executable memory | | | Automatically if supported |
| EXCEPTION_HANDLIN | Ask all SimExceptions raised during execution to be handled by the SimOS | | tracing | |
| FAST_MEMORY | Use SimFastMemory for memory storage | | | |
| FAST_REGISTERS | Use SimFastMemory for register storage | | fastpath | |
| INITIALIZE_ZERO_R | Treat the initial value of registers as zero instead of unconstrained symbolic | unicorn | tracing | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| KEEP_IP_SYMBOLIC | Don't try to concretize successor states with symbolic instruction pointers | | | |
| KEEP_MEMORY_READS | In abstract memory, handle failed loads by returning a DCIS? | | | |
| LAZY_SOLVES | Don't check satisfiability until absolutely necessary | | | |
| MEMORY_SYMBOLIC_B | Maintain a mapping of symbolic variable to which memory address it "really" corresponds to, at the paged memory level? | | | |
| NO_SYMBOLIC_JUMP_ | Do not attempt to flatten symbolic-ip successors into discrete targets | | fastpath | |
| NO_SYMBOLIC_SYSCA | Do not attempt to flatten symbolic-syscall-number successors into discrete targets | | fastpath | |
| OPTIMIZE_IR | Use LibVEX's optimization | common_options | all | |
| REGION_MAPPING | Maintain a mapping of symbolic variable to which memory region it corresponds to, at the abstract memory level | | static | |
| REPLACEMENT_SOLVE | Enable SolverReplacement | | | |
| REVERSE_MEMORY_HA | Maintain a mapping from AST hash to which addresses it is present in | | | |
| REVERSE_MEMORY_NA | Maintain a mapping from symbolic variable name to which addresses it is present in, required for memory.replace_all | | static | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| SIMPLIFY_CONSTRAI | Run added constraints through z3's simplifcation | | | |
| SIMPLIFY_EXIT_GUA | Run branch guards through z3's simplification | | | |
| SIMPLIFY_EXIT_STA | Perform simplification on all successor states generated | | | |
| SIMPLIFY_EXIT_TAR | Run jump/call/branch targets through z3's simplification | | | |
| SIMPLIFY_EXPRS | Run the results of IR expressions through z3's simplification | | | |
| SIMPLIFY_MEMORY_R | Run the results of memory reads through z3's simplification | | | |
| SIMPLIFY_MEMORY_W | Run values stored to memory through z3's simplification | simplification, common_options | symbolic, symbolic_approxim tracing | |
| SIMPLIFY_REGISTER | Run values read from registers through z3's simplification | | | |
| SIMPLIFY_REGISTER | Run values written to registers through z3's simplification | simplification, common_options | symbolic, symbolic_approxim tracing | |
| SIMPLIFY_RETS | Run values returned from SimProcedures through z3's simplification | | | |
| STRICT_PAGE_ACCES | Raise a SimSegfaultException when attempting to interact with memory in a way not permitted by the current permissions | | tracing | |
| SUPER_FASTPATH | Only execute the last four instructions of each block | | | |
| SUPPORT_FLOATING_ | When disabled, throw an UnsupportedIROpError when encountering floating point operations | common_options | all | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| SYMBOLIC | Enable constraint solving? | symbolic | symbolic, symbolic_approxim fastpath | |
| SYMBOLIC_INITIAL_ | make state.solver.Unconstrained return a symbolic value instead of zero | symbolic | all | |
| SYMBOLIC_TEMPS | Treat each IR temporary as a symbolic variable; treat stores to them as constraint addition | | | |
| SYMBOLIC_WRITE_AD | Allow writes with symbolic addresses to be processed by concretization strategies; when disabled, only allow for variables annotated with the "multiwrite" annotation | | | |
| TRACK_CONSTRAINTS | When disabled, don't keep any constraints added to the state | symbolic | all | |
| TRACK_CONSTRAINT_ | Keep a SimAction for each constraint added | refs | | |
| TRACK_JMP_ACTIONS | Keep a SimAction for each jump or branch | refs | | |
| TRACK_MEMORY_ACTI | Keep a SimAction for each memory read and write | refs | | |
| TRACK_MEMORY_MAPP | Keep track of which pages are mapped into memory and which are not | common_options | all | |
| TRACK_OP_ACTIONS | Keep a SimAction for each IR operation | | fastpath | |
| TRACK_REGISTER_AC | Keep a SimAction for each register read and write | refs | | |
| TRACK_SOLVER_VARI | Maintain a listing of all the variables in all the constraints in the solver | | | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| TRACK_TMP_ACTIONS | Keep a SimAction for each temporary variable read and write | refs | | |
| TRUE_RET_EMULATIC | With DO_RET_EMULATION, add fake returns with guard condition true instead of false | | static | |
| UNDER_CONSTRAINED | Enable under-constrained symbolic execution | | | |
| UNICORN | Use unicorn engine to execute symbolically when data is concrete | unicorn | tracing | Oppologist |
| UNICORN_AGGRESSIV | Concretize any register variable unicorn tries to access | | | Oppologist |
| UNICORN_HANDLE_TR | CGC: handle the transmit syscall without leaving unicorn | unicorn | tracing | |
| UNICORN_SYM_REGS_ | Attempt to stay in unicorn even in the presence of symbolic registers by checking that the tainted registers are unused at every step | unicorn | tracing | |
| UNICORN_THRESHOLD | Concretize variables if they prevent unicorn from executing too often | | | |
| UNICORN_TRACK_BBL | Keep state.history.bbl_addrs up to date when using unicorn | unicorn | tracing | |
| UNICORN_TRACK_STA | Track a list of the stack pointer's value at each block in state.scratch.stack_pointer_lis | unicorn | | |
| UNICORN_ZEROPAGE_ | Prevent unicorn from mapping the zero page into memory | | | |

continues on next page

Table 1 – continued from previous page

| Option name | Description | Sets | Modes | Implicit adds |
|---|---|---|---|---|
| UNINITIALIZED_ACC | Broken/unused? | | | |
| UNSUPPORTED_BYPAS | When using the resilience options, return zero instead of an unconstrained symbol | | | |
| USE_SIMPLIFIED_CC | Use a "simplified" set of ccalls optimized for specific cases | | static | |
| USE_SYSTEM_TIMES | In library functions and syscalls and hardware instructions accessing clock data, retrieve the real value from the host system. | | tracing | |
| VALIDATE_APPROXIM | Debug: When performing approximations, ensure that the approximation is sound by calling into z3 | | | |
| ZERO_FILL_UNCONSTI | Make the value of memory read from an uninitialized address zero instead of an unconstrained symbol | | tracing | |

## 9.4 CTF Challenge Examples

angr is very often used in CTFs. These are example scripts resulting from that use, mostly from Shellphish but also from many others.

### 9.4.1 ReverseMe example: HackCon 2016 - angry-reverser

Script author: Stanislas Lejay (github: @P1kachu)

Script runtime: ~31 minutes

Here is the binary and the script

### 9.4.2 ReverseMe example: SecurityFest 2016 - fairlight

Script author: chuckleberryfinn (github: @chuckleberryfinn)

Script runtime: ~20 seconds

A simple reverse me that takes a key as a command line argument and checks it against 14 checks. Possible to solve the challenge using angr without reversing any of the checks.

Here is the binary and the script

### 9.4.3 ReverseMe example: DEFCON Quals 2016 - baby-re

Authors David Manouchehri (github: @Manouchehri), Stanislas Lejay (github: @P1kachu) and Audrey Dutcher (github: @rhelmot).

Script runtime: 10 sec

Here is the binary and the script

### 9.4.4 ReverseMe example: Google CTF - Unbreakable Enterprise Product Activation (150 points)

Script 0 author: David Manouchehri (github: @Manouchehri)

Script runtime: 4.5 sec

Script 1 author: Adam Van Prooyen (github: @docileninja)

Script runtime: 6.7 sec

A Linux binary that takes a key as a command line argument and checks it against a series of constraints.

Challenge Description:

> We need help activating this product – we've lost our license key :(
>
> You're our only hope!

Here are the binary and scripts: script 0, script_1

### 9.4.5 ReverseMe example: EKOPARTY CTF - Fuckzing reverse (250 points)

Author: Adam Van Prooyen (github: @docileninja)

Script runtime: 29 sec

A Linux binary that takes a team name as input and checks it against a series of constraints.

Challenge Description:

> Hundreds of conditions to be meet, will you be able to surpass them?

Both sample binaries and the script are located here and additional information be found at the author's write-up.

### 9.4.6 ReverseMe example: WhiteHat Grant Prix Global Challenge 2015 - Re400

Author: Fish Wang (github: @ltfish)

Script runtime: 5.5 sec

A Windows binary that takes a flag as argument, and tells you if the flag is correct or not.

"I have to patch out some checks that are difficult for angr to solve (e.g., it uses some bytes of the flag to decrypt some data, and see if those data are legit Windows APIs). Other than that, angr works really well for solving this challenge."

The binary and the script.

### 9.4.7 ReverseMe example: EKOPARTY CTF 2015 - rev 100

Author: Fish Wang (github: @ltfish)

Script runtime: 5.5 sec

This is a painful challenge to solve with angr. I should have done things in a smarter way.

Here is the binary and the script.

### 9.4.8 ReverseMe example: ASIS CTF Finals 2015 - fake

Author: Fish Wang (github: @ltfish)

Script runtime: 1 min 57 sec

The solution is pretty straight-forward.

The binary and the script.

### 9.4.9 ReverseMe example: Defcamp CTF Qualification 2015 - Reversing 100

Author: Fish Wang (github: @ltfish)

angr solves this challenge with almost zero user-interference.

See the script and the binary.

### 9.4.10 ReverseMe example: Defcamp CTF Qualification 2015 - Reversing 200

Author: Fish Wang (github: @ltfish)

angr solves this challenge with almost zero user-interference. Veritesting is required to retrieve the flag promptly.

The script and the binary. It takes a few minutes to run on my laptop.

### 9.4.11 ReverseMe example: MMA CTF 2015 - HowToUse

Author: Audrey Dutcher (github: @rhelmot)

We solved this simple reversing challenge with angr, since we were too lazy to reverse it or run it in Windows. The resulting script shows how we grabbed the flag out of the DLL.

### 9.4.12 CrackMe example: MMA CTF 2015 - SimpleHash

Author: Chris Salls (github: @salls)

This crackme is 95% solvable with angr, but we did have to overcome some difficulties. The script describes the difficulties that were encountered and how we worked around them. The binary can be found here.

### 9.4.13 ReverseMe example: FlareOn 2015 - Challenge 10

Author: Fish Wang (github: @ltfish)

angr acts as a binary loader and an emulator in solving this challenge. I didn't have to load the driver onto my Windows box.

The script demonstrates how to hook at arbitrary program points without affecting the intended bytes to be executed (a zero-length hook). It also shows how to read bytes out of memory and decode as a string.

By the way, here is the link to the intended solution from FireEye.

### 9.4.14 ReverseMe example: FlareOn 2015 - Challenge 2

Author: Chris Salls (github: @salls)

This reversing challenge is simple to solve almost entirely with angr, and a lot faster than trying to reverse the password checking function. The script is here

### 9.4.15 ReverseMe example: 0ctf 2016 - momo

Author: Fish Wang (github: @ltfish), ocean (github: @ocean1)

This challenge is a movfuscated binary. To find the correct password after exploring the binary with Qira it is possible to understand how to find the places in the binary where every character is checked using capstone and using angr to load the binary and brute-force the single characters of the flag. Be aware that the script is really slow. Runtime: > 1 hour.

### 9.4.16 CrackMe example: 9447 CTF 2015 - Reversing 330, "nobranch"

Author: Audrey Dutcher (github: @rhelmot)

angr cannot currently solve this problem natively, as the problem is too complex for z3 to solve. Formatting the constraints to z3 a little differently allows z3 to come up with an answer relatively quickly. (I was asleep while it was solving, so I don't know exactly how long!) The script for this is here and the binary is here.

### 9.4.17 CrackMe example: ais3_crackme

Author: Antonio Bianchi, Tyler Nighswander

ais3_crackme has been developed by Tyler Nighswander (tylerni7) for ais3 summer school. It is an easy crackme challenge, checking its command line argument.

### 9.4.18 ReverseMe: Modern Binary Exploitation - CSCI 4968

Author: David Manouchehri (GitHub @Manouchehri)

This folder contains scripts used to solve some of the challenges with angr. At the moment it only contains the examples from the IOLI crackme suite, but eventually other solutions will be added.

### 9.4.19 CrackMe example: Android License Check

Author: Bernhard Mueller (GitHub @b-mueller)

A native binary for Android/ARM that validates a license key passed as a command line argument. It was created for the symbolic execution tutorial in the OWASP Mobile Testing Guide.

## 9.5 Changelog

This lists the *major* changes in angr. Tracking minor changes are left as an exercise for the reader :-)

### 9.5.1 angr 9.1

- (#2961) Refactored SimCC to support passing and returning structs and arrays by value
- (#2964) Functions from the knowledge base may now be pretty-printed, showing colors and reference arrows
- Improved `import angr` speed substantially
- (#2948) RDA's `dep_graph` can now be used to track dependencies between temporaries, constants, guard conditions, and function calls - if you want it!
- (#2929) Basic support for structs with bitfields in SimType
- There's a decompiler now

### 9.5.2 angr 9.0

- Switched to a new versioning scheme: major.minor.build_id

### 9.5.3 angr 8.19.7.25

- (#1503) Implement necessary helpers and information storage for call pretty printing
- (#1546) Add a new state option MEMORY_FIND_STRICT_SIZE_LIMIT
- (#1548) SimProcedure.static_exits: Allow providing name hints
- (cle#177) Use Enums for Symbol Types
- (cle#193) Add support for "named regions"
- (claripy#151) Implement operator precedence in claripy op rendering
- Added support for interaction recording in angr-management
- Several new simprocedure implementations
- Substantial imporvments to our CFG

### 9.5.4 angr 8.19.4.5

- (#1234) Massive improvements to CFG recovery for ARM and ARM cortex-m binaries.
- (#1416) Added support for analyzing Java programs via the Soot IR, including the ability to analyze interplay between Java code and JNI libraries. This branch was two years old!
- (#1427) Added a MemoryWatcher exploration technique to take action when the system is running out of RAM. Thanks @bannsec.
- (#1432) Added a `state.heap` plugin which manages the heap (with pluggable heap schemes!) and provides malloc functionality. Thanks @tgduckworth.
- Speed improvements for using the VEX engine and working with concrete data.
- Added SimLightRegisters, an alternate registers plugin that eliminates the abstraction of the register file for performance improvements at the cost of removing all instrumentability.
- `version__` variable has been added to all modules.
- The `stack_base` kwarg for `call_state` is not broken for the first time ever
- https://github.com/python/cpython/pull/11384

### 9.5.5 angr 8.19.2.4

- (#1279) Support C++ function name demangling via itanium-demangler. Thanks @fmagin.
- (#1283) `security_cookie` is initialized for SimWindows. Thanks @zeroSteiner.
- (#1298) Introduce `SimData`. It's a cleaner interface to deal with data imports in CLE – especially for those data entries that are not imported because of missing or unloaded libraries. This commit fixes long-standing issues #151 and #693.
- (#1299, #1300, #1301, #1313, #1314, #1315, #1336, #1337, #1343, ...) Multiple CFGFast-related improvements and bug fixes.
- (#1332) `UnresolvableTarget` is now split into two classes: `UnresolvableJumpTarget` and `UnresolvableCallTarget`. Thanks @Kyle-Kyle.

- (#1382) Add a preliminary implementation of angr decompiler. Give it a try! `p = angr.Project("cfg_loop_unrolling", auto_load_libs=False); p.analyses.CFG(); print(p.analyses.Decompiler(p.kb.functions['test_func']).codegen.text)`.
- (#1421) `SimAction`s now have incrementing IDs. Thanks @bannsec.
- (#1408) `ANA`, angr's old identity-aware serialization backend, has been removed. Instead of non-obvious serialization behavior, all angr objects should now be pickleable. If one is not, please file an issue. For use-cases that require identity-awareness (i.e., deduplicating ASTs across states serialized at different times), an `angr.vaults` module has been introduced.
- Added a facility to synchronize state between angr and a running target a la avatar2
- Changed unconstrained registers/memory warning to be less obnoxious and contain useful information. Also added `SYMBOL_FILL_UNCONSTRAINED_REGISTERS` and `SYMBOL_FILL_UNCONSTRAINED_MEMORY` state options to silence them.

### 9.5.6 angr 8.18.10.25

- The IDA backend for CLE has been removed. It has been broken for quite some time, but now it has been disabled for your own safety.
- Surveyors have been removed! Finally! This is thanks to @danse-macabre who contributed an Exploration Technique for the Slicecutor. Backwards slicing has now been brought out of the angr dark ages.
- SimCC can now be initialized with a string containing C function prototype in its `func_ty` argument
- Similarly, Callable can now be run with its arguments instantiated from a string containing C expressions
- Tracer has been substantially refactored - it will now handle more kinds of desyncs, ASLR slides, and is much more friendly for hacking. We will be continuing to improve it!
- The Oppologist and Driller have been refactored to play nice with other exploration techniques
- SimProcedure continuations now have symbols in the externs object, so `describe_addr` will work on them. Additionally, the representation for SimProcedure (appearing in `history.descriptions` and `project._sim_procedures` among other places) has been improved to show this information.

### 9.5.7 angr 8.18.10.5

Largely a bugfix release, but with a few bonus treats:

- API documentation has been rewritten for Exploration Technique. It should be much easier to use now.
- Simulation Manager will throw an error if you pass incorrect keyword arguments (??? why was it like this)
- The `save_unconstrained` flag of Simulation Manager is now on by default
- If a step produces only unsatisfiable states, they will appear in the `'unsat'` stash regardless of the `save_unsat` setting, since this usually indicates a bug. Add `unsat` to the `auto_drop` parameter to restore the old behavior.

### 9.5.8 angr 8.18.10.1

Welcome to angr 8! The biggest change for this major version bump is the transition to Python 3. You can read about this, as well as a few other breaking changes, in the *Migrating to angr 8*.

- Switch to Python 3
- Refactor to Clemory to clean up the API and speed things up drastically
- Remove `object.symbols_by_addr` (dict) and add `object.symbols` (sorted list); add `fuzzy` parameter to `loader.find_symbol`

- CFGFast is much, much faster now. CFGAccurate has been renamed to CFGEmulated.
- Support for avx2 unpack instructions, courtesy of D. J. Bernstein
- Removed support for immutable simulation managers
- angr will now show you a warning when using uninitialized memory or registers
- angr will now NOT show you a warning if you have a capstone 3.x install unless you're actually interacting with the relevant missing parts
- Many, many, many bug fixes

### 9.5.9 angr 7.8.7.1

- Remove `LoopLimiter` and `DFG`.
- (#1063) `CFGAccurate` can now leverage indirect jump resolvers to resolve indirect jumps.

### 9.5.10 angr 7.8.6.23

- (PyVEX!#134) We now recognize LDMDB r11, {xxx, pc} as a ret instruction for ARM.
- (#1053) CFGFast spends less time running next_pos_with_sort_not_in(), thus it runs faster on large binaries.
- (#1080) Jump table resolvers now support resolving ARM jump tables.
- (#1081, together with the PyVEX commit 61efbdcf6303a936aa3de35011d2d1e3fe5fdea5) The memory footprint of CFGFast is noticeably smaller, especially on large binaries (over 10 MB in size).
- (#1034) Concretizing a SimFile with unconstrained size can no longer run you out of memory.
- Other minor changes and bug fixes.

### 9.5.11 angr 7.8.6.16

- The modeling of file system is refactored.
- (#808) Add a new class Control flow blanket (CFBlanket) to support generating a linear view of a control flow graph.
- (#863) Add support to AIL, the new angr intermediate language (still pretty WIP though). Merged in several static analyses (reaching definition analysis, VEX-to-AIL translation, redundant assignment elimination, code region identification, control flow structuring, etc.) that support the development of decompilation in the near future.
- (#888) SimulationManager is extensively refactored and cleaned up.
- (#892) Keystone is integrated. You can assemble instructions inside angr now.
- (#897) A new class `PluginHub` is added. Plugins (analyses, engines) are refactored to be based on `PluginHub`.
- (#899) Support of bidirectional mapping between syscall numbers and syscalls.
- (#925, #941, #942) A bunch of library function prototypes (including glibc) are added to angr.
- (#953) Fix the issue where evaluating the jump target of a jump table that contains many entries (e.g., > 512) is extremely slow.
- (#964) State options are now stored in insances of SimStateOptions. `state.options` is no longer a set of strings.
- (#973) Add two new exploration techniques: Stochastic and unique.
- (#996) SimType structs are now much easier to use.
- (#998) Add a new state option `PRODUCE_ZERODIV_SUCCESSORS` to generate divide-by-zero successors.

- Speed improvements and bug fixes in CFG generation (CFGFast and CFGAccurate).

### 9.5.12 angr 7.8.2.21

- Refactor of how syscall handling and SimSyscallLibrary work - it is now possible to handle syscalls using multiple ABIs in the same process
- Added syscall name-number mappings from all linux ABIs, parsed from gdb
- Add `ManualMergepoint` exploration technique for when veritesting is too mysterious for your tastes
- Add `LoopSeer` exploration technique for managing loops during symbolic exploration (credit @tyb0807)
- Add `ProxyTechnique` exploration technique for easily composing simple lambda-based instrumentations (credit @danse-macabre)

### 9.5.13 angr 7.7.12.16

- You can now tell where the variables implicitly created by angr come from! `state.solver.BVS` now can take a `key` parameter, which describes its meaning in relation to the emulated environment. You can then use `state.solver.get_variables(...)` and `state.solver.describe_variables(...)` to map tags and ASTs to and from each other. Check out the API docs!
- The SimOS for a project is now a public property - `project.simos` instead of `project._simos`. Additionally, the SimOS code structure has been shuffled around a bit - it's now a subpackage instead of a submodule.
- The core components of Tracer and Driller have been refactored into Exploration Techniques and integrated into angr proper, so you can now follow instruction traces without installing another repository! (credit @tyb0807)
- Archinfo now contains a `byte_width` parameter and angr supports emulation of platforms with non-octet bytes, lord help us
- Upgraded to networkx 2 (credit @tyb0807)
- Hopefully installation issues with capstone should be fixed FOREVER
- Minor fixes to gender

### 9.5.14 angr 7.7.9.8

Welcome to angr 7! We worked long and hard all summer to make this release the best ever. It introduces several breaking changes, so for a quick guide on the most common ways you'll need to update your scripts, take a look at the *Migrating to angr 7*.

- SimuVEX has been removed and its components have been integrated into angr
- Path has been removed and its components have been integrated into SimState, notably the new `history` state plugin
- PathGroup has been renamed to SimulationManager
- SimState and SimProcedure now have a reference to their parent Project, though it is verboten to use it in anything other than an append-only fashion
- A new class SimLibrary is used to track SimProcedure and metadata corresponding to an individual shared library
- Several CLE interfaces have been refactored up for consistency
- Hook has been removed. Hooking is now done with individual SimProcedure instances, which are shallow-copied at execution time for thread-safety.
- The `state.solver` interface has been cleaned up drastically

These are the major refactor-y points. As for the improvements:

- Greatly improved support for analyzing 32 bit windows binaries (partial credit @schieb)
- Unicorn will now stop for stop points and breakpoints in the middle of blocks (credit @bennofs)
- The processor flags for a state can now be accessed through `state.regs.eflags` on x86 and `state.regs.flags` on ARM (partial credit @tyb0807)
- Fledgling support for emulating exception handling. Currently the only implementation of this is support for Structured Exception Handling on Windows, see `angr.SimOS.handle_exception` for details
- Fledgling support for runtime library loading by treating the CLE loader as an append-only interface, though only implemented for windows. See `cle.Loader.dynamic_load` and `angr.procedures.win32.dynamic_loading` for details.
- The knowledge base has been refactored into a series of plugins similar to SimState (credit @danse-macabre)
- The testcase-based function identifier we wrote for CGC has been integrated into angr as the Identifier analysis
- Improved support for writing custom VEX lifters

### 9.5.15 angr 6.7.6.9

- angr: A static data-flow analysis framework has been introduced, and implemented as part of the `ForwardAnalysis` class. Additionally, a few exemplary data-flow analyses, like `VariableRecovery` and `VariableRecoveryFast`, have been implemented in angr.
- angr: We introduced the notion of *variable* to the angr world. Now a VariableManager is available in the knowledge base. Variable information can be recovered by running a variable recovery analysis. Currently the variable information recovered for each function is still pretty coarse. More updates to it will arrive soon.
- angr: Fix a bug in the topological sorting in `CFGUtils`, which resulted in suboptimal graph node ordering after sorting.
- SimuVEX: `LAZY_SOLVES` is no longer enabled by default during symbolic execution. It's still there if it's wanted, but it just caused confusion when on by default.
- SimuVEX: Thanks to @ekilmer, a few new libc SimProcedures are added.
- SimuVEX: The default memory model has been refactored for expandability. Custom pages can now be created (derive the simuvex.storage.ListPage class) and used instead of the default page classes to implement custom memory behavior for specific pages. The user-friendly API for this is pending the next release.
- angr-management: Implemented our own graph layout and edge routing algorithm. We do not rely on grandalf anymore.
- angr-management: Added support for displaying variable information for operands.
- angr-management: Added support for highlighting dependent operands when an operand is highlighted.

### 9.5.16 angr 6.7.3.26

Building off of the engine changes from the last release, we have begun to extend angr to other architectures. AVR and MSP430 are in progress. In the meantime, subwire has created a reference implementation of BrainFuck support in angr, done two different ways! Check out angr-platforms for more info!

- We have rebased our fork of VEX on the latest master branch from Valgrind (as of 2 months ago, at least...). We have also submitted our patches to VEX to upstream, so we should be able to stop maintaining a fork pretty soon.
- The way we interact with VEX has changed substantially, and should speed things up a bit.
- Loading sets of binaries with many import symbols has been sped up

- Many, many improvements to angr-management, including the switch away from enaml to using pyside directly.

### 9.5.17 angr 6.7.1.13

For the last month, we have been working on a major refactor of the angr to change the way that angr reasons about the code that it analyzes. Until now, angr has been bound to the VEX intermediate representation to lift native code, supporting a wide range of architectures but not being very expandable past them. This release represents the ground work for what we call translation and execution engines. These engines are independent backends, pluggable into the angr framework, that will allow angr to reason about a wide range of targets. For now, we have restructured the existing VEX and Unicorn Engine support into this engine paradigm, but as we discuss in our blog post, the plan is to create engines to enable angr's reasoning of Java bytecode and source code, and to augment angr's environment support through the use of external dynamic sandboxes.

For now, these changes are mostly internal. We have attempted to maintain compatibility for end-users, but those building systems atop angr will have to adapt to the modern codebase. The following are the major changes:

- simuvex: we have introduced SimEngine. SimEngine is a base class for abstractions over native code. For example, angr's VEX-specific functionality is now concentrated in SimEngineVEX, and new engines (such as SimEngineLLVM) can be implemented (even outside of simuvex itself) to support the analysis of new types of code.
- simuvex: as part of the engines refactor, the SimRun class has been eliminated. Instead of different subclasses of SimRun that would be instantiated from an input state, engines each have a `process` function that, from an input state, produces a SimSuccessors instance containing lists of different successor states (normal, unsat, unconstrained, etc) and any engine-specific artifacts (such as the VEX statements. Take a look at `successors.artifacts`).
- simuvex: `state.mem[x:] = y` now *requires* a type for storage (for example `state.mem[x:].dword = y`).
- simuvex: the way of calling inline SimProcedures has been changed. Now you have to create a SimProcedure, and then call `execute()` on it and pass in a program state as well as the arguments.
- simuvex: accessing registers through `SimRegNameView` (like `state.regs.eax`) always triggers SimInspect breakpoints and creates new actions. Now you can access a register by prefixing its name with an underscore (e.g. `state.regs._eax` or `state._ip`) to avoid triggering breakpoints or creating actions.
- angr: the way hooks work has slightly changed, though is backwards-compatible. The new angr.Hook class acts as a wrapper for hooks (SimProcedures and functions), keeping things cleaner in the `project._sim_procedures` dict.
- angr: we have deprecated the keyword argument `max_size` and changed it to to `size` in the `angr.Block` constructor (i.e., the argument to `project.factory.block` and more upstream methods (`path.step`, `path_group.step`, etc).
- angr: we have deprecated `project.factory.sim_run` and changed it to to `project.factory.successors`, and it now generates a `SimSuccessors` object.
- angr: `project.factory.sim_block` has been deprecated and replaced with `project.factory.successors(default_engine=True)`.
- angr: angr syscalls are no longer hooks. Instead, the syscall table is now in `project._simos.syscall_table`. This will be made "public" after a usability refactor. If you were using `project.is_hooked(addr)` to see if an address has a related SimProcedure, now you probably want to check if there is a related syscall as well (using `project._simos.syscall_table.get_by_addr(addr) is not None`).
- pyvex: to support custom lifters to VEX, pyvex has introduced the concept of backend lifters. Lifters can be written in pure Python to produce VEX IR, allowing for extendability of angr's VEX-based analyses to other hardware architectures.

As usual, there are many other improvements and minor bugfixes.

- claripy: support `unsat_core()` to get the core of unsatness of constraints. It is in fact a thin wrapper of the `unsat_core()` function provided by Z3. Also a new state option `CONSTRAINT_TRACKING_IN_SOLVER` is added to SimuVEX. That state option must be enabled if you want to use `unsat_core()` on any state.
- simuvex: `SimMemory.load()` and `SimMemory.store()` now takes a new parameter `disable_actions`. Setting it to True will prevent any SimAction creation.
- angr: CFGFast has a better support for ARM binaries, especially for code in THUMB mode.
- angr: thanks to an improvement in SimuVEX, CFGAccurate now uses slightly less memory than before.
- angr: `len()` on path `trace` or `addr_trace` is made much faster.
- angr: Fix a crash during CFG generation or symbolic execution on platforms/architectures with no syscall defined.
- angr: as part of the refactor, `BackwardSlicing` is temporarily disabled. It will be re-enabled once all DDG-related refactor are merged to master.

Additionally, packaging and build-system improvements coordinated between the angr and Unicorn Engine projects have allowed angr's Unicorn support to be built on Windows. Because of this, `unicorn` is now a dependency for `simuvex`.

Looking forward, angr is poised to become a program analysis engine for binaries *and more*!

### 9.5.18 angr 5.6.12.3

It has been over a month since the last release 5.6.10.12. Again, we've made some significant changes and improvements on the code base.

- angr: Labels are now stored in KnowledgeBase.
- angr: Add a new analysis: `Disassembly`. The new Disassembly analysis provides an easy-to-use interface to render assembly of functions.
- angr: Fix the issue that `ForwardAnalysis` may prematurely terminate while there are still un-processed jobs.
- angr: Many small improvements and bug fixes on `CFGFast`.
- angr: Many small improvements and bug fixes on `VFG`. Bring back widening support. Fix the issue that `VFG` may not terminate under certain cases. Implement a new graph traversal algorithm to have an optimal traversal order. Allow state merging at non-merge-points, which allows faster convergence.
- angr-management: Display a progress during initial CFG recovery.
- angr-management: Display a "Load binary" window upon binary loading. Some analysis options can be adjusted there.
- angr-management: Disassembly view: Edge routing on the graph is improved.
- angr-management: Disassembly view: Support starting a new symbolic execution task from an arbitrary address in the program.
- angr-management: Disassembly view: Support renaming of function names and labels.
- angr-management: Disassembly view: Support "Jump to address".
- angr-management: Disassembly view: Display resolved and unresolved jump targets. All jump targets are double-clickable.
- SimuVEX: Move region mapping from `SimAbstractMemory` to `SimMemory`. This will allow an easier conversion between `SimAbstractMemory` and `SimSymbolicMemory`, which is to say, conversion between symbolic states and static states is now possible.

- SimuVEX & claripy: Provide support for `unsat_core` in Z3. It returns a set of constraints that led to unsatness of the constraint set on the current state.
- archinfo: Add a new Boolean variable `branch_delay_slot` for each architecture. It is set to True on MIPS32.

### 9.5.19 angr 5.6.8.22

Major point release! An incredible number of things have changed in the month run-up to the Cyber Grand Challenge.

- Integration with Unicorn Engine supported for concrete execution. A new SimRun type, SimUnicorn, may step through many basic blocks at once, so long as there is no operation on symbolic data. Please use our fork of unicorn engine, which has many patches applied. All these patches are pending merge into upstream.
- Lots of improvements and bug fixes to CFGFast. Rumors are angr's CFG was only "optimized" for x86-64 binaries (which is really because most of our test cases are compiled as 64-bit ELFs). Now it is also "optimized" for x86 binaries :) (editor's note: angr is built with cross-architecture analysis in mind. CFG construction is pretty much the only component which has architecture-specific behavior.)
- Lots of improvements to the VFG analysis, including speed and accuracy. However, there is still a lot to be done.
- Lots of speed optimizations in general - CFGFast should be 3-6x faster under CPython with much less memory usage.
- Now data dependence graph gives you a real dependence graph between variable definitions. Try `data_graph` and `simplified_data_graph` on a DDG object!
- New state option `simuvex.o.STRICT_PAGE_ACCESS` will cause a `SimSegfaultError` to be raised whenever the guest reads/writes/executes memory that is either unmapped or doesn't have the appropriate permissions.
- Merging of paths (as opposed to states) is performed in a much smarter way.
- The behavior of the `support_selfmodifying_code` project option is changed: Before, this would allow the state to be used as a fallback source of instruction bytes when no backer from CLE is available. Now, this option makes instruction lifting use the state as the source of bytes always. When the option is disabled and execution jumps outside the normal binary, the state will be used automatically.
- *Actually* support self-modifying code - if a basic block of code modifies itself, the block will be re-lifted before the next instruction starts.
- Syscalls are handled differently now - Before you would see a SimRun for a syscall helper, now you'll just see a SimProcedure for the given syscall. Additionally, each syscall has its own address in a "syscalls segment", and syscalls are treated as jumps to this segment. This simplifies a lot of things analysis-wise.
- CFGAccurate accepts a `base_graph` keyword to its constructor, e.g. `CFGFast().graph`, or even `.graph` of a function, to use as a base for analysis.
- New fast memory model for cases where symbolic-addressed reads and writes are unlikely.
- Conflicts between the `find` and `avoid` parameters to the Explorer otiegnqwvk are resolved correctly. (credit clslgrnc)
- New analysis `StaticHooker` which hooks library functions in unstripped statically linked binaries.
- `Lifter` can be used without creating an angr Project. You must manually specify the architecture and bytestring in calls to `.lift()` and `.fresh_block()`. If you like, you can also specify the architecture as a parameter to the constructor and omit it from the lifting calls.
- Add two new analyses developed for the CGC (mostly as examples of doing static analysis with angr): Reassembler and BinaryOptimizer.

### 9.5.20 angr 4.6.6.28

In general, there have been enormous amounts of speed improvements in this release. Depending on the workload, angr should run about twice as fast. Aside from this, there have also been many submodule-specific changes:

#### angr

Quite a few changes and improvements are made to `CFGFast` and `CFGAccurate` in order to have better and faster CFG recovery. The two biggest changes in `CFGFast` are jump table resolution and data references collection, respectively. Now `CFGFast` resolves indirect jumps by default. You may get a list of indirect jumps recovered in `CFGFast` by accessing the `indirect_jumps` attribute. For many cases, it resolves the jump table accurately. Data references collection is still in alpha mode. To test data references collection, just pass `collect_data_references=True` when creating a fast CFG, and access the `memory_data` attribute after the CFG is constructed.

CFG recovery on ARM binaries is also improved.

A new paradigm called an "otiegnqwvk", or an "exploration technique", allows the packaging of special logic related to path group stepping.

#### SimuVEX

Reads/writes to the x87 fpu registers now work correctly - there is special logic that rotates a pointer into part of the register file to simulate the x87 stack.

With the recent changes to Claripy, we have configured SimuVEX to use the composite solver by default. This should be transparent, but should be considered if strange issues (or differences in behavior) arise during symbolic execution.

#### Claripy

Fixed a bug in claripy where `div__` was not always doing unsigned division, and added new methods `SDiv` and `SMod` for signed division and signed remainder, respectively.

Claripy frontends have been completely rewritten into a mixin-centric solver design. Basic frontend functionality (i.e., calling into the solver or dealing with backends) is handled by frontends (in `claripy.frontends`), and additional functionality (such as caching, deciding when to simplify, etc) is handled by frontend mixins (in `claripy.frontend_mixins`). This makes it considerably easier to customize solvers to your specific needE. For examples, look at `claripy/solver.py`.

Alongside the solver rewrite, the composite solver (which splits constraints into independent constraint sets for faster solving) has been immensely improved and is now functional and fast.

### 9.5.21 angr 4.6.6.4

Syscalls are no longer handled by `simuvex.procedures.syscalls.handler`. Instead, syscalls are now handled by `angr.SimOS.handle_syscall()`. Previously, the address of a syscall SimProcedure is the address right after the syscall instruction (e.g. `int 80h`), which collides with the real basic block starting at that address, and is very confusing. Now each syscall SimProcedure has its own address, just as a normal SimProcedure. To support this, there is another region mapped for the syscall addresses, `Project._syscall_obj`.

Some refactoring and bug fixes in `CFGFast`.

Claripy has been given the ability to handle *annotations* on ASTs. An annotation can be used to customize the behavior of some backends without impacting others. For more information, check the docstrings of `claripy.Annotation` and `claripy.Backend.apply_annotation`.

### 9.5.22 angr 4.6.5.25

New state constructor - `call_state`. Comes with a refactor to `SimCC`, a refactor to `callable`, and the removal of `PathGroup.call`. All these changes are thoroughly documented, in `angr/docs/advanced-topics/structured_data.md`

Refactor of `SimType` to make it easier to use types - they can be instantiated without a SimState and one can be added later. Comes with some usability improvements to SimMemView. Also, there's a better wrapper around PyCParser for generating SimType instances from c declarations and definitions. Again, thoroughly documented, still in the structured data doc.

`CFG` is now an alias to `CFGFast` instead of `CFGAccurate`. In general, `CFGFast` should work under most cases, and it's way faster than `CFGAccurate`. We believe such a change is necessary, and will make angr more approachable to new users. You will have to change your code from `CFG` to `CFGAccurate` if you are relying on specific functionalities that only exist in `CFGAccurate`, for example, context-sensitivity and state-preserving. An exception will be raised by angr if any parameter passed to `CFG` is only supported by `CFGAccurate`. For more detailed explanation, please take a look at the documentation of `angr.analyses.CFG`.

### 9.5.23 angr 4.6.3.28

PyVEX has a structural overhaul. The `IRExpr`, `IRStmt`, and `IRConst` modules no longer exist as submodules, and those module names are deprecated. Use `pyvex.expr`, `pyvex.stmt`, and `pyvex.const` if you need to access the members of those modules.

The names of the first three parameters to `pyvex.IRSB` (the required ones) have been changed. If you were passing the positional args to IRSB as keyword args, consider switching to positional args. The order is `data`, `mem_addr`, `arch`.

The optional parameter `sargc` to the `entry_state` and `full_init_state` constructors has been removed and replaced with an `argc` parameter. `sargc` predates being able to have claripy ASTs independent from a solver. The new system is to pass in the exact value, ast or integer, that you'd like to have as the guest program's arg count.

CLE and angr can now accept file-like streams, that is, objects that support `stream.read()` and `stream.seek()` can be passed in wherever a filepath is expected.

Documentation is much more complete, especially for PyVEX and angr's symbolic execution control components.

### 9.5.24 angr 4.6.3.15

There have been several improvements to claripy that should be transparent to users:

- There's been a refactoring of the VSA StridedInterval classes to fix cases where operations were not sound. Precision might suffer as a result, however.
- Some general speed improvements.
- We've introduced a new backend into claripy: the ReplacementBackend. This frontend generates replacement sets from constraints added to it, and uses these replacement sets to increase the precision of VSA. Additionally, we have introduced the HybridBackend, which combines this functionality with a constraint solver, allowing for memory index resolution using VSA.

angr itself has undergone some improvements, with API changes as a result:

- We are moving toward a new way to store information that angr has recovered about a program: the knowledge base. When an analysis recovers some truth about a program (i.e., "there's a basic block at 0x400400", or "the block at 0x400400 has a jump to 0x400500"), it gets stored in a knowledge-base. Analysis that used to store data (currently, the CFG) now store them in a knowledge base and can *share* the global knowledge base of the project, now accessible via `project.kb`. Over time, this knowledge base will be expanded in the course of any analysis or symbolic execution, so angr is constantly learning more information about the program it is analyzing.
- A forward data-flow analysis framework (called ForwardAnalysis) has been introduced, and the CFG was rewritten on top of it. The framework is still in alpha stage - expect more changes to be made. Documentation and

more details will arrive shortly. The goal is to refactor other data-flow analysis, like CFGFast, VFG, DDG, etc. to use ForwardAnalysis.

- We refactored the CFG to a) improve code readability, and b) eliminate some bad designs that linger due to historical reasons.

### 9.5.25 angr 4.5.12.?

Claripy has a new manager for backends, allowing external backends (i.e., those implemented by other modules) to be used. The result is that `claripy.backend_concrete` is now `claripy.backends.concrete`, `claripy.backend_vsa` is now `claripy.backends.vsa`, and so on.

### 9.5.26 angr 4.5.12.12

Improved the ability to recover from failures in instruction decoding. You can now hook specific addresses at which VEX fails to decode with `project.hook`, even if those addresses are not the beginning of a basic block.

### 9.5.27 angr 4.5.11.23

This is a pretty beefy release, with over half of claripy having been rewritten and major changes to other analyses. Internally, Claripy has been unified – the VSA mode and symbolic mode now work on the same structures instead of requiring structures to be created differently. This opens the door for awesome capabilities in the future, but could also result in unexpected behavior if we failed to account for something.

Claripy has had some major interface changes:

- claripy.BV has been renamed to claripy.BVS (bit-vector symbol). It can now create bitvectors out of strings (i.e., claripy.BVS(0x41, 8) and claripy.BVS("A") are identical).
- state.BV and state.BVV are deprecated. Please use state.se.BVS and state.se.BVV.
- BV.model is deprecated. If you're using it, you're doing something wrong, anyways. If you really need a specific model, convert it with the appropriate backend (i.e., claripy.backend_concrete.convert(bv)).

There have also been some changes to analyses:

- Interface: CFG argument `keep_input_state` has been renamed to `keep_state`. With this option enabled, both input and final states are kept.
- Interface: Two arguments `cfg_node` and `stmt_id` of `BackwardSlicing` have been deprecated. Instead, `BackwardSlicing` takes a single argument, `targets`. This means that we now support slicing from multiple sources.
- Performance: The speed of CFG recovery has been slightly improved. There is a noticeable speed improvement on MIPS binaries.
- Several bugs have been fixed in DDG, and some sanity checks were added to make it more usable.

And some general changes to angr itself:

- StringSpec is deprecated! You can now pass claripy bitvectors directly as arguments.

## 9.6 Migrating to angr 9.1

angr 9.1 is here!

### 9.6.1 Calling Conventions and Prototypes

The main change motivating angr 9.1 is this large refactor of SimCC. Here are the breaking changes:

#### SimCCs can no longer be customized

If you were using the `sp_delta`, `args`, or `ret_val` parameters to SimCC, you should use the new class `SimCCUsercall`, which lets (requires) you to be explicit about the locations of each argument.

#### Passing SimTypes is now mandatory

Every method call on SimCC which interacts with typed data now requires a SimType to be passed in. Previously, the use of `is_fp` and `size` was optional, but now these parameters will no longer be accepted and a `SimType` will be required.

This has some fairly non-intuitive consequences - in order to accommodate more esoteric calling conventions (think: passing large structs by value via an "invisible reference") you have to specify a function's return type before you can extract any of its arguments.

Additionally, some non-cc interfaces, such as `call_state` and `callable` and `SimProcedure.call()`, now *require* a prototype to be passed to them. You'd be surprised how many bugs we found in our own code from enforcing this requirement!

#### PointerWrapper has a new parameter

Imagine you're passing something into a function which has a parameter of type `char*`. Is this a pointer to a single char or a pointer to an array of chars? The answer changes how we typecheck the values you pass in. If you're passing a PointerWrapper wrapping a large value which should be treated as an array of chars, you should construct your pointerwrapper as `PointerWrapper(foo, buffer=True)`. The buffer argument to PointerWrapper now instructs SimCC to treat the data to be serialized as an array of the child type instead of as a scalar.

#### `func_ty -> prototype`

Every usage of the name func_ty has been replaced with the name prototype. This was done for consistency between the static analysis code and the dynamic FFI.

## 9.7 Migrating to angr 8

angr has moved from Python 2 to Python 3! We took this opportunity of a major version bump to make a few breaking API changes that improve quality-of-life.

### 9.7.1 What do I need to know for migrating my scripts to Python 3?

To begin, just the standard py3k changes, the relevant parts of which we'll rehash here as a reference guide:

- Strings and bytestrings
  - Strings are now unicode by default, a new `bytes` type holds bytestrings
  - Bytestring literals can be constructed with the b prefix, like `b'ABCD'`
  - Conversion between strings and bytestrings happens with `.encode()` and `.decode()`, which use utf-8 as a default. The `latin-1` codec will map byte values to their equivalent unicode codepoints
  - The `ord()` and `chr()` functions operate on strings, not bytestrings
  - Enumerating over or indexing into bytestrings produces an unsigned 8 bit integer, not a 1-byte bytestring
  - Bytestrings have all the string manipulation functions present on strings, including `join`, `upper/lower`, `translate`, etc

  - `hex` and `base64` are no longer string encoding codecs. For hex, use `bytes.fromhex()` and `bytes.hex()`. For base64 use the `base64` module.

- Builtin functions
  - `print` and `exec` are now builtin functions instead of statements
  - Many builtin functions previously returning lists now return iterators, such as `map`, `filter`, and `zip`. `reduce` is no longer a builtin; you have to import it from `functools`.

- Numbers
  - The `/` operator is explicitly floating-point division, the `//` operator is expliclty integer division. The magic functions for overriding these ops are `truediv__` and `floordiv__`
  - The int and long types have been merged, there is only int now

- Dictionary objects have had their `.iterkeys`, `.itervalues`, and `.iteritems` methods removed, and then non-iter versions have been made to return efficient iterators

- Comparisons between objects of very different types (such as between strings and ints) will raise an exception

In terms of how this has affected angr, any string that represents data from the emulated program will be a bytestring. This means that where you previously said `state.solver.eval(x, cast_to=str)` you should now say `cast_to=bytes`. When creating concrete bitvectors from strings (including implicitly by just making a comparison against a string) these should be bytestrings. If they are not they will be utf-8 converted and a warning will be printed. Symbol names should be unicode strings.

For division, however, ASTs are strongly typed so they will treat both division operators as the kind of division that makes sense for their type.

### 9.7.2 Clemory API changes

The memory object in CLE (project.loader.memory, not state.memory) has had a few breaking API changes since the bytes type is much nicer to work with than the py2 string for this specific case, and the old API was an inconsistent mess.

| Before | After |
|---|---|
| `memory.read_bytes(addr, n) -> list[str]` | `memory.load(addr, n) -> bytes` |
| `memory.write_bytes(addr, list[str])` | `memory.store(addr, bytes)` |
| `memory.get_byte(addr) -> str` | `memory[addr] -> int` |
| `memory.read_addr_at(addr) -> int` | `memory.unpack_word(addr) -> int` |
| `memory.write_addr_at(addr, value) -> int` | `memory.pack_word(addr, value)` |
| `memory.stride_repr -> list[(start, end, str)]` | `memory.backers() -> iter[(start, bytearray)]` |

Additionally, `pack_word` and `unpack_word` now take optional `size`, `endness`, and `signed` parameters. We have also added `memory.pack(addr, fmt, *data)` and `memory.unpack(addr, fmt)`, which take format strings for use with the `struct` module.

If you were using the `cbackers` or `read_bytes_c` functions, the conversion is a little more complicated - we were able to remove the split notion of "backers" and "updates" and replaced all backers with bytearrays that we mutate, so we can work directly with the backer objects. The `backers()` function iterates through all bottom-level backer objects and their start addresses. You can provide an optional address to the function, and it will skip over all backers that end before that address.

Here is some sample code for producing a C-pointer to a given address:

```
import cffi, cle
ffi = cffi.FFI()
ld = cle.Loader('/bin/true')

addr = ld.main_object.entry
try:
    backer_start, backer = next(ld.memory.backers(addr))
except StopIteration:
    raise Exception("not mapped")

if backer_start > addr:
    raise Exception("not mapped")

cbacker = ffi.from_buffer(backer)
addr_pointer = cbacker + (addr - backer_start)
```

You should not have to use this if you aren't passing the data to a native library - the normal load methods should now be more than fast enough for intensive use.

### 9.7.3 CLE symbols changes

Previously, your mechanisms for looking up symbols by their address were `loader.find_symbol()` and `object.symbols_by_addr`, where there was clearly some overlap. However, `symbols_by_addr` stayed because it was the only way to enumerate symbols in an object. This has changed! `symbols_by_addr` is deprecated and here is now `object.symbols`, a sorted list of Symbol objects, to enumerate symbols in a binary.

Additionally, you can now enumerate all symbols in the entire project with `loader.symbols`. This change has also enabled us to add a `fuzzy` parameter to `find_symbol` (returns the first symbol before the given address) and make the output of `loader.describe_addr` much nicer (shows offset from closest symbol).

### 9.7.4 Deprecations and name changes

- All parameters in cle that started with `custom_` - so, `custom_base_addr`, `custom_entry_point`, `custom_offset`, `custom_arch`, and `custom_ld_path` - have had the `custom_` removed from the beginning of their names.
- All the functions that were deprecated more than a year ago (at or before the angr 7 release) have been removed.
- `state.se` has been deprecated. You should have been using `state.solver` for the past few years.
- Support for immutable simulation managers has been removed. So far as we're aware, nobody was actually using this, and it was making debugging a pain.

## 9.8 Migrating to angr 7

The release of angr 7 introduces several departures from long-standing angr-isms. While the community has created a compatibility layer to give external code written for angr 6 a good chance of working on angr 7, the best thing to do is to port it to the new version. This document serves as a guide for this.

### 9.8.1 SimuVEX is gone

angr versions up through angr 6 split the program analysis into two modules: `simuvex`, which was responsible for analyzing the effects of a single piece of code (whether a basic block or a SimProcedure) on a program state, and `angr`, which aggregated analyses of these basic blocks into program-level analysis such as control-flow recovery, symbolic execution, and so forth. In theory, this would encourage for the encapsulation of block-level analyses, and allow other

program analysis frameworks to build upon `simuvex` for their needs. In practice, no one (to our knowledge) used `simuvex` without `angr`, and the separation introduced frustrating limitations (such as not being able to reference the history of a state from a SimInspect breakpoint) and duplication of code (such as the need to synchronize data from `state.scratch` into `path.history`).

Realizing that SimuVEX wasn't a usable independent package, we brainstormed about merging it into angr and further noticed that this would allow us to address the frustrations resulting from their separation.

All of the SimuVEX concepts (SimStates, SimProcedures, calling conventions, types, etc) have been migrated into angr. The migration guide for common classes is bellow:

| Before | After |
|---|---|
| simuvex.SimState | angr.SimState |
| simuvex.SimProcedure | angr.SimProcedure |
| simuvex.SimEngine | angr.SimEngine |
| simuvex.SimCC | angr.SimCC |

And for common modules:

| Before | After |
|---|---|
| simuvex.s_cc | angr.calling_conventions |
| simuvex.s_state | angr.sim_state |
| simuvex.s_procedure | angr.sim_procedure |
| simuvex.plugins | angr.state_plugins |
| simuvex.engines | angr.engines |
| simuvex.concretization_strategies | angr.concretization_strategies |

Additionally, `simuvex.SimProcedures` has been renamed to `angr.SIM_PROCEDURES`, since it is a global variable and not a class. There have been some other changes to its semantics, see the section on SimProcedures for details.

### 9.8.2 Removal of angr.Path

In angr, a Path object maintained references to a SimState and its history. The fact that the history was separated from the state caused a lot of headaches when trying to analyze states inside a breakpoint, and caused overhead in synchronizing data from the state to its history.

In the new model, a state's history is maintained in a SimState plugin: `state.history`. Since the path would now simply point to the state, we got rid of it. The mapping of concepts is roughly as follows:

| Before | After |
|---|---|
| path | state |
| path.state | state |
| path.history | state.history |
| path.callstack | state.callstack |
| path.trace | state.history.descriptions |
| path.addr_trace | state.history.bbl_addrs |
| path.jumpkinds | state.history.jumpkinds |
| path.guards | state.history.jump_guards |
| path.targets | state.history.jump_targets |
| path.actions | state.history.actions |
| path.events | state.history.events |
| path.recent_actions | state.history.recent_actions |
| path.reachable | state.history.reachable() |

An important behavior change about `path.actions` and `path.recent_actions` - actions are no longer tracked by default. If you would like them to be tracked again, please add `angr.options.refs` to your state.

#### Path Group -> Simulation Manager

Since there are no paths, there cannot be a path group. Instead, we have a Simulation Manager now (we recommend using the abbreviation "simgr" in places you were previously using "pg"), which is exactly the same as a path group except it holds states instead of paths. You can make one with `project.factory.simulation_manager(...)`.

#### Errored Paths

Before, error resilience was handled at the path level, where stepping a path that caused an error would return a subclass of Path called ErroredPath, and these paths would be put in the `errored` stash of a path group. Now, error resilience is handled at the simulation manager level, and any state that throws an error during stepping will be wrapped in an ErrorRecord object, which is *not* a subclass of SimState, and put into the `errored` list attribute of the simulation manager, which is *not* a stash.

An ErrorRecord object has attributes for `.state` (the initial state that caused the error), `.error` (the error that was thrown), and `.traceback` (the traceback from the error). To debug these errors you can call `.debug()`.

These changes are because we were uncomfortable making a subclass of SimState, and the ErrorRecord class then has sufficiently different semantics from a normal state that it cannot be placed in a stash.

### 9.8.3 Changes to SimProcedures

The most noticeable difference from the old version to the new version is that the catalog of built-in simprocedures are no longer organized strictly according to which library they live in. Now, they are organized according to which *standards* they conform to, which helps with re-using procedures between different libraries. For instance, the old `SimProcedures['libc.so.6']` has been split up between `SIM_PROCEDURES['libc']`, `SIM_PROCEDURES['posix']`, and `SIM_PROCEDURES['glibc']`, depending on what specifications each function conforms to. This allows us to reuse the `libc` catalog in `msvcrt.dll` and the MUSL libc, for example.

In order to group SimProcedures together by libraries, we have introduced a new abstraction called the SimLibrary, the definitions for which are stored in `angr.procedures.definitions`. Each SimLibrary object stores information about a single shared library, and can contain SimProcedure implementations, calling convention information, and type information. SimLibraries are scraped from the filesystem at import time, just like SimProcedures, and placed into `angr.SIM_LIBRARIES`.

Syscalls are now categorized through a subclass of SimLibrary called SimSyscallLibrary. The API for managing syscalls through SimOS has been changed - check the API docs for the SimUserspace class.

One important implication of this change is that if you previously used a trick where you changed one of the SimProcedures present in the `SimProcedures` dict in order to change which SimProcedures would be used to hook over library functions by default, this will no longer work. Instead of `SimProcedures[lib][func_name] = proc`, you now need to say `SIM_LIBRARIES[lib].add(func_name, proc)`. But really you should just be using `hook_symbol` anyway.

### 9.8.4 Changes to hooking

The `Hook` class is gone. Instead, we now can hook with individual instances of SimProcedure objects, as opposed to just the classes. A shallow copy of the SimProcedure will be made at runtime to preserve thread safety.

So, previously, where you would have done `project.hook(addr, Hook(proc, ...))` or `project.hook(addr, proc)`, you can now do `project.hook(addr, proc(...))`. In order to use simple functions as hooks, you can either say `project.hook(addr, func)` or decorate the declaration of your function with `@project.hook(addr)`.

Having simprocedures as instances and letting them have access to the project cleans up a lot of other hacks that were present in the codebase, mostly related to the `self.call(...)` SimProcedure continuation system. It is no longer required to set `IS_FUNCTION = True` if you intend to use `self.call()` while writing a SimProcedure, and each call-return target you use will have a unique address associated with it. These addresses will be allocated lazily, which does have the side effect of making address allocation nondeterministic, sometimes based on dictionary-iteration order.

### 9.8.5 Changes to loading

The `hook_symbol` method will no longer attempt to redo relocations for the given symbol, instead just hooking directly over the address of the symbol in whatever library it comes from. This speeds up loading substantially and ensures more consistent behavior for when mixing and matching native library code and SimProcedure summaries.

The angr externs object has been moved into CLE, which will ALWAYS make sure that every dependency is resolved to something, never left unrelocated. Similarly, CLE provides the "kernel object" used to provide addresses for syscalls now.

| Before | After |
|---|---|
| `project._extern_obj` | `loader.extern_object` |
| `project._syscall_obj` | `loader.kernel_object` |

Several properties and methods have been renamed in CLE in order to maintain a more consistent and explicit API. The most common changes are listed below:

| Before | After |
|---|---|
| `loader.whats_at()` | `loader.describe_addr` |
| `loader.addr_belongs_to_object()` | `loader.find_object_containing()` |
| `loader.find_symbol_name()` | `loader.find_symbol().name` |
| whatever the hell you were doing before to look up a symbol | `loader.find_symbol(name or addr)` |
| `loader.find_module_name()` | `loader.find_object_containing().provides` |
| `loader.find_symbol_got_entry()` | `loader.find_relevant_relocations()` |
| `loader.main_bin` | `loader.main_object` |
| `anything.get_min_addr()` | `anything.min_addr` |
| `symbol.addr` | `symbol.linked_addr` |

### 9.8.6 Changes to the solver interface

We cleaned up the menagerie of functions present on `state.solver` (if you're still referring to it as `state.se` you should stop) and simplified it into a cleaner interface:

- `solver.eval(expression)` will give you one possible solution to the given expression.
- `solver.eval_one(expression)` will give you the solution to the given expression, or throw an error if more than one solution is possible.
- `solver.eval_upto(expression, n)` will give you up to n solutions to the given expression, returning fewer than n if fewer than n are possible.
- `solver.eval_atleast(expression, n)` will give you n solutions to the given expression, throwing an error if fewer than n are possible.
- `solver.eval_exact(expression, n)` will give you n solutions to the given expression, throwing an error if fewer or more than are possible.
- `solver.min(expression)` will give you the minimum possible solution to the given expression.
- `solver.max(expression)` will give you the maximum possible solution to the given expression.

Additionally, all of these methods can take the following keyword arguments:

- `extra_constraints` can be passed as a tuple of constraints. These constraints will be taken into account for this evaluation, but will not be added to the state.
- `cast_to` can be passed a data type to cast the result to. Currently, this can only be `str`, which will cause the method to return the byte representation of the underlying data. For example, `state.solver.eval(state.solver.BVV(0x41424344, 32, cast_to=str)` will return `"ABCD"`.

CHAPTER

# TEN

# API REFERENCE

**class** angr.**BP**

Bases: object

A breakpoint.

**__init__**(*when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

**Parameters**

- **when** (When)
- **enabled** (*bool*)
- **condition** (*Callable[[SimState], bool] | None*)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

**check**(*state*, *when*)

Checks state *state* to see if the breakpoint should fire.

**Parameters**

- **state** (*SimState*) – The state.
- **when** (*When*) – Whether the check is happening before or after the event.

**Return type**
bool

**Returns**
A boolean representing whether the checkpoint should fire.

**fire**(*state*)

Trigger the breakpoint.

**Parameters**
**state** (*SimState*) – The state.

angr.**BP_IPDB**(*state*)

**Return type**
None

**Parameters**
**state** (SimState)

angr.**BP_IPYTHON**(*state*)

**Return type**
None

**Parameters**
**state** (SimState)

**class** angr.**Analysis**

Bases: object

This class represents an analysis on the program.

**Variables**

- **project** – The project for this analysis.
- **kb** (KnowledgeBase) – The knowledgebase object.
- **_progress_callback** – A callback function for receiving the progress of this analysis. It only takes one argument, which is a float number from 0.0 to 100.0 indicating the current progress.
- **_show_progressbar** (*bool*) – If a progressbar should be shown during the analysis. It's independent from _progress_callback.
- **_progressbar** (*progress.Progress*) – The progress bar object.

**project:** *Project*

**kb:** *KnowledgeBase*

**errors:** list[*AnalysisLogEntry*] = []

**named_errors:** defaultdict[str, list[*AnalysisLogEntry*]] = {}

**log:** list

**property ram_usage:** float

Return the current RAM usage of the Python process, in bytes. The value is updated at most once per second.

**exception** angr.**AngrAIError**

Bases: *AngrError*

**exception** angr.**AngrAnalysisError**

Bases: *AngrError*

**exception** angr.**AngrAnnotatedCFGError**

Bases: *AngrError*

**exception** angr.**AngrAssemblyError**

Bases: *AngrError*

**exception** angr.**AngrBackwardSlicingError**

Bases: *AngrError*

**exception** angr.**AngrBladeError**

Bases: *AngrError*

**exception** angr.**AngrBladeSimProcError**

Bases: *AngrBladeError*

**exception** angr.**AngrCFGError**

Bases: *AngrError*

**exception** angr.**AngrCallableError**

Bases: *AngrSurveyorError*

**exception** angr.**AngrCallableMultistateError**

Bases: *AngrCallableError*

**exception** angr.**AngrCorruptDBError**

Bases: *AngrDBError*

**exception** angr.**AngrDBError**

Bases: *AngrError*

**exception** angr.**AngrDDGError**

Bases: *AngrAnalysisError*

**exception** angr.**AngrDataGraphError**

Bases: *AngrAnalysisError*

**exception** angr.**AngrDecompilationComplexityError**

Bases: *AngrDecompilationError*

Raised when a function exceeds one of the decompiler's complexity limits. The decompiler gives up before starting the expensive work instead of running for an unbounded amount of time.

**Variables**

- **limit_name** – Name of the decompilation option that sets the limit, e.g. "max_function_blocks".
- **limit** – The configured limit.
- **actual** – The measured size that exceeded the limit.
- **func_addr** – Address of the offending function, if known.
- **unit** – What `actual` counts, e.g. "basic blocks".

**exception** angr.**AngrDecompilationError**

Bases: *AngrError*

**exception** angr.**AngrDelayJobNotice**

Bases: *AngrForwardAnalysisError*

**exception** angr.**AngrDirectorError**

Bases: *AngrExplorationTechniqueError*

**exception** angr.**AngrError**

Bases: Exception

**exception** angr.**AngrExitError**

Bases: *AngrError*

**exception** angr.**AngrExplorationTechniqueError**

Bases: *AngrError*

**exception** angr.**AngrExplorerError**

Bases: *AngrExplorationTechniqueError*

**exception** angr.**AngrForwardAnalysisError**

Bases: *AngrError*

**exception** angr.**AngrIncompatibleDBError**

Bases: *AngrDBError*

**exception** angr.**AngrIncongruencyError**

Bases: *AngrAnalysisError*

**exception** angr.**AngrInvalidArgumentError**

Bases: *AngrError*

**exception** angr.**AngrJobMergingFailureNotice**

Bases: *AngrForwardAnalysisError*

**exception** angr.**AngrJobWideningFailureNotice**

Bases: *AngrForwardAnalysisError*

**exception** angr.**AngrLifterError**

Bases: *AngrError*

**exception** angr.**AngrLoopAnalysisError**

Bases: *AngrAnalysisError*

**exception** angr.**AngrMissingTypeError**

Bases: *AngrTypeError*

**exception** angr.**AngrNoPluginError**

Bases: *AngrError*

**exception** angr.**AngrPathError**

Bases: *AngrError*

**exception** angr.**AngrRuntimeError**

Bases: RuntimeError

**exception** angr.**AngrSimOSError**

Bases: *AngrError*

**exception** angr.**AngrSkipJobNotice**

Bases: *AngrForwardAnalysisError*

**exception** angr.**AngrSurveyorError**

Bases: *AngrError*

**exception** angr.**AngrSyscallError**

Bases: *AngrError*

**exception** angr.**AngrTracerError**

Bases: *AngrExplorationTechniqueError*

**exception** angr.**AngrTypeError**

Bases: *AngrError*, TypeError

**exception** angr.**AngrUnsupportedSyscallError**

Bases: *AngrSyscallError*, *SimProcedureError*, *SimUnsupportedError*

**exception** angr.**AngrVFGError**

Bases: *AngrError*

**exception** angr.**AngrVFGRestartAnalysisNotice**

Bases: *AngrVFGError*

**exception** angr.**AngrValueError**

Bases: *AngrError*, ValueError

**exception** angr.**AngrVaultError**

Bases: *AngrError*

**class** angr.**Blade**

Bases: object

Blade is a light-weight program slicer that works with networkx DiGraph containing CFGNodes. It is meant to be used in angr for small or on-the-fly analyses.

**__init__**(*graph*, *dst_run*, *dst_stmt_idx*, *direction='backward'*, *project=None*, *cfg=None*, *ignore_sp=False*, *ignore_bp=False*, *ignored_regs=None*, *max_level=3*, *base_state=None*, *stop_at_calls=False*, *cross_insn_opt=False*, *max_predecessors=10*, *include_imarks=True*, *control_dependence=True*)

**Parameters**

- **graph** (DiGraph) – A graph representing the control flow graph. Note that it does not take angr.analyses.CFGEmulated or angr.analyses.CFGFast.
- **dst_run** (int) – An address specifying the target SimRun.
- **dst_stmt_idx** (int) – The target statement index. -1 means executing until the last statement.
- **direction** (str) – 'backward' or 'forward' slicing. Forward slicing is not yet supported.
- **project** (*angr.Project*) – The project instance.
- **cfg** (*angr.analyses.CFGBase*) – the CFG instance. It will be made mandatory later.
- **ignore_sp** (bool) – Whether the stack pointer should be ignored in dependency tracking. Any dependency from/to stack pointers will be ignored if this options is True.
- **ignore_bp** (bool) – Whether the base pointer should be ignored or not.
- **max_level** (int) – The maximum number of blocks that we trace back for.
- **stop_at_calls** (bool) – Limit slicing within a single function. Do not proceed when encounters a call edge.
- **include_imarks** (bool) – Should IMarks (instruction boundaries) be included in the slice.
- **control_dependence** (bool) – Whether to consider control dependencies. If True, the temps controlling conditional exits will be added to the tainting set.
- **max_predecessors** (*int*)

**Returns**

None

**property slice**

**dbg_repr**(*arch=None*)

class angr.**Block**

Bases: *Serializable*

Represents a basic block in a binary or a program.

**BLOCK_MAX_SIZE = 4096**

**__init__**(*addr*, *project=None*, *arch=None*, *size=None*, *max_size=None*, *byte_string=None*, *thumb=False*, *backup_state=None*, *extra_stop_points=None*, *opt_level=None*, *num_inst=None*, *traceflags=0*, *strict_block_end=None*, *collect_data_refs=False*, *cross_insn_opt=True*, *load_from_ro_regions=False*, *const_prop=False*, *initial_regs=None*, *skip_stmts=False*)

**Parameters**
**arch** (*Arch | None*)

**arch**

**addr**

**thumb**

**size**

**pp**(***kwargs*)

**set_initial_regs**()

**static reset_initial_regs**()

**property vex:** IRSB | *IRSB*

**property vex_nostmt**

**property disassembly:** *DisassemblerBlock*

Provide a disassembly object using whatever disassembler is available

**property capstone:** *CapstoneBlock*

**property pcode:** *PCodeBlock*

**property codenode**

**property bytes:** *bytes* | None

**property instructions:** int

**property instruction_addrs**

class angr.**Emulator**

Bases: object

Emulator is a utility that adapts an angr *SuccessorsEngine* to a more user-friendly interface for concrete execution. It only supports concrete execution.

Saftey: This class is not thread-safe. It should only be used in a single-threaded context. It can be safely shared between multiple threads, provided that only one thread is using it at a time.

**__init__**(*engine*, *init_state*)

**Parameters**

- **engine** (*SuccessorsEngine*) – The *SuccessorsEngine* to use for emulation.
- **init_state** (*SimState*) – The initial state to use for emulation.

**property state: *SimState***

The current state of the emulator.

**property breakpoints: set[int]**

The set of currently set breakpoints.

**add_breakpoint**(*addr*)

Add a breakpoint at the given address.

**Parameters**
**addr** (int) – The address to set the breakpoint at.

**Return type**
None

**remove_breakpoint**(*addr*)

Remove a breakpoint at the given address, if present.

**Parameters**
**addr** (int) – The address to remove the breakpoint from.

**Return type**
None

**run**(*num_inst=None*)

Execute the emulator.

**Return type**
*EmulatorStopReason*

**Parameters**
**num_inst** (*int | None*)

**class** angr.**EmulatorStopReason**

Bases: Enum

Enum representing the reason for stopping the emulator.

**INSTRUCTION_LIMIT = 'instruction_limit'**

**BREAKPOINT = 'breakpoint'**

**NO_SUCCESSORS = 'no_successors'**

**MEMORY_ERROR = 'memory_error'**

**FAILURE = 'failure'**

**EXIT = 'exit'**

class angr.ExplorationTechnique

Bases: object

An ExplorationTechnique is a set of hooks for a simulation manager that assists in the implementation of new techniques in symbolic exploration.

Any number of these methods may be overridden by a subclass. To use an exploration technique, call `simgr.use_technique` with an *instance* of the technique.

__init__()

setup(*simgr*)

Perform any initialization on this manager you might need to do.

**Parameters**

**simgr** (*angr.SimulationManager*) – The simulation manager to which you have just been added

step(*simgr*, *stash='active'*, ***kwargs*)

Hook the process of stepping a stash forward. Should call `simgr.step(stash, **kwargs)` in order to do the actual processing.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **stash** (*str*)

filter(*simgr*, *state*, ***kwargs*)

Perform filtering on which stash a state should be inserted into.

If the state should be filtered, return the name of the stash to move the state to. If you want to modify the state before filtering it, return a tuple of the stash to move the state to and the modified state. To defer to the original categorization procedure, return the result of `simgr.filter(state, **kwargs)`

If the user provided a `filter_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

selector(*simgr*, *state*, ***kwargs*)

Determine if a state should participate in the current round of stepping. Return True if the state should be stepped, and False if the state should not be stepped. To defer to the original selection procedure, return the result of `simgr.selector(state, **kwargs)`.

If the user provided a `selector_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

step_state(*simgr*, *state*, ***kwargs*)

Determine the categorization of state successors into stashes. The result should be a dict mapping stash names to the list of successor states that fall into that stash, or None as a stash name to use the original stash name.

If you would like to directly work with a *SimSuccessors* object, you can obtain it with `simgr.successors(state, **kwargs)`. This is not recommended, as it denies other hooks the opportunity to

look at the successors. Therefore, the usual technique is to call `simgr.step_state(state, **kwargs)` and then mutate the returned dict before returning it yourself.

..note:: This takes precedence over the *filter* hook - *filter* is only applied to states returned from here in the None stash.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**successors**(*simgr*, *state*, ***kwargs*)

Perform the process of stepping a state forward, returning a SimSuccessors object.

To defer to the original succession procedure, return the result of `simgr.successors(state, **kwargs)`. Be careful about not calling this method (e.g. calling `project.factory.successors` manually) as it denies other hooks the opportunity to instrument the step. Instead, you can mutate the kwargs for the step before calling the original, and mutate the result before returning it yourself.

If the user provided a `successor_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**complete**(*simgr*)

Return whether or not this manager has reached a "completed" state, i.e. `SimulationManager.run()` should halt.

This is the one hook which is *not* subject to the nesting rules of hooks. You should *not* call `simgr.complete`, you should make your own decision and return True or False. Each of the techniques' completion checkers will be called and the final result will be compted with `simgr.completion_mode`.

**Parameters**
**simgr** (*angr.SimulationManager*)

**class** angr.**KnowledgeBase**

Bases: `object`

Represents a "model" of knowledge about an artifact.

Contains things like a CFG, data references, etc.

**functions:** *FunctionManager*

**comments:** *Comments*

**bookmarks:** *Bookmarks*

**variables:** *VariableManager*

**defs:** *KeyDefinitionManager*

**cfgs:** *CFGManager*

**types:** *TypesStore*

**propagations:** *PropagationManager*

**xrefs:** *XRefManager*

**decompilations:** *StructuredCodeManager*

**obfuscations:** *Obfuscations*

**rtdb:** *RuntimeDb*

**__init__**(*project*, *obj=None*, *name=None*)

**property callgraph**

**property unresolved_indirect_jumps**

**property resolved_indirect_jumps**

**has_plugin**(*name*)

**get_plugin**(*name*)

**register_plugin**(*name*, *plugin*)

**release_plugin**(*name*)

**get_knowledge**(*requested_plugin_cls*)

Type inference safe method to request a knowledge base plugin Explicitly passing the type of the requested plugin achieves two things: 1. Every location using this plugin can be easily found with an IDE by searching explicit references to the type 2. Basic type inference can deduce the result type and properly type check usages of it

If there isn't already an instance of this class None will be returned to make it clear to the caller that there is no existing knowledge of this type yet. The code that initially creates this knowledge should use the *register_plugin* method to register the initial knowledge state :type requested_plugin_cls: type[TypeVar(K, bound= *KnowledgeBasePlugin*)] :param requested_plugin_cls:

**Return type**
Optional[TypeVar(K, bound= *KnowledgeBasePlugin*)]

**Returns**
Instance of the requested plugin class or null if it is not a known plugin

**Parameters**
**requested_plugin_cls** (type*[K]*)

**request_knowledge**(*requested_plugin_cls*)

**Return type**
TypeVar(K, bound= *KnowledgeBasePlugin*)

**Parameters**
**requested_plugin_cls** (type*[K]*)

**class** angr.**LLMClient**

Bases: object

A client for interacting with LLMs via pydantic-ai. Used by the decompiler to suggest improved variable names, function names, and types.

**__init__**(*model*, *api_key=None*, *api_base=None*, *max_tokens=4096*, *temperature=0.0*)

**Parameters**

- **model** (*str*)

- **api_key** (*str | None*)
- **api_base** (*str | None*)
- **max_tokens** (*int*)
- **temperature** (*float*)

**infer_provider**(*provider*)

Infer the provider from the provider name.

**Return type**
Provider[Any]

**Parameters**
**provider** (*str*)

**classmethod from_env**()

Create an LLMClient from environment variables.

Uses ANGR_LLM_MODEL (required), ANGR_LLM_API_KEY (optional), and ANGR_LLM_API_BASE (optional). Returns None if ANGR_LLM_MODEL is not set.

**Return type**
*LLMClient* | None

**completion**(*messages*, ***kwargs*)

Call the LLM with the given messages and return the response text.

**Return type**
str

**Parameters**
**messages** (*list[dict[str, str]]*)

**completion_structured**(*messages*, *output_type*, *raise_exc=False*, ***kwargs*)

Call the LLM with the given messages and return a validated Pydantic model. Returns None if the call fails.

**Parameters**

- **raise_exc** (bool) – If True, exceptions are propagated to the caller instead of being caught.
- **messages** (*list[dict[str, str]]*)
- **output_type** (*type[T]*)

**Return type**
Optional[TypeVar(T)]

**completion_json**(*messages*, *raise_exc=False*, ***kwargs*)

Call the LLM and parse the response as JSON. Strips markdown code fences if present. Returns None on parse failure. Kept for backwards compatibility; prefer completion_structured().

**Parameters**

- **raise_exc** (bool) – If True, exceptions are propagated to the caller instead of being caught.
- **messages** (*list[dict[str, str]]*)

**Return type**
dict | None

class angr.PTChunk

Bases: *Chunk*

A chunk, inspired by the implementation of chunks in ptmalloc. Provides a representation of a chunk via a view into the memory plugin. For the chunk definitions and docs that this was loosely based off of, see glibc malloc/malloc.c, line 1033, as of commit 5a580643111ef6081be7b4c7bd1997a5447c903f. Alternatively, take the following link. https://sourceware.org/git/?p=glibc.git;a=blob;f=malloc/malloc.c;h=67cdfd0ad2f003964cd0f7dfe3bcd85ca98528a7;hb=5a580643111ef6081be7b4c7bd1997a5447c903f#l1033

**Variables**

- **base** – the location of the base of the chunk in memory
- **state** – the program state that the chunk is resident in
- **heap** – the heap plugin that the chunk is managed by

__init__(*base*, *sim_state*, *heap=None*)

set_size(*size*, *is_free=None*)

Use this to set the size on a chunk. When the chunk is new (such as when a free chunk is shrunk to form an allocated chunk and a remainder free chunk) it is recommended that the is_free hint be used since setting the size depends on the chunk's freeness, and vice versa.

**Parameters**

- **size** – size of the chunk
- **is_free** – boolean indicating the chunk's freeness

set_prev_freeness(*is_free*)

Sets (or unsets) the flag controlling whether the previous chunk is free.

**Parameters**

**is_free** – if True, sets the previous chunk to be free; if False, sets it to be allocated

is_prev_free()

Returns a concrete state of the flag indicating whether the previous chunk is free or not. Issues a warning if that flag is symbolic and has multiple solutions, and then assumes that the previous chunk is free.

**Returns**

True if the previous chunk is free; False otherwise

prev_size()

Returns the size of the previous chunk, masking off what would be the flag bits if it were in the actual size field. Performs NO CHECKING to determine whether the previous chunk size is valid (for example, when the previous chunk is not free, its size cannot be determined).

next_chunk()

Returns the chunk immediately following (and adjacent to) this one, if it exists.

**Returns**

The following chunk, or None if applicable

prev_chunk()

Returns the chunk immediately prior (and adjacent) to this one, if that chunk is free. If the prior chunk is not free, then its base cannot be located and this method raises an error.

**Returns**

If possible, the previous chunk; otherwise, raises an error

**fwd_chunk**()

Returns the chunk following this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the forward chunk; otherwise, raises an error

**bck_chunk**()

Returns the chunk backward from this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the backward chunk; otherwise, raises an error

**exception** angr.**PathUnreachableError**

Bases: *AngrPathError*

**class** angr.**PointerWrapper**

Bases: object

**__init__**(*value*, *buffer=False*)

**class** angr.**Project**

Bases: object

This is the main class of the angr module. It is meant to contain a set of binaries and the relationships between them, and perform analyses on them.

**Parameters**

- **thing** – The path to the main executable object to analyze, or a CLE Loader object.
- **default_analysis_mode** – The mode of analysis to use by default. Defaults to 'symbolic'.
- **ignore_functions** – A list of function names that, when imported from shared libraries, should never be stepped into in analysis (calls will return an unconstrained value).
- **use_sim_procedures** – Whether to replace resolved dependencies for which simprocedures are available with said simprocedures.
- **exclude_sim_procedures_func** – A function that, when passed a function name, returns whether or not to wrap it with a simprocedure.
- **exclude_sim_procedures_list** – A list of functions to *not* wrap with simprocedures.
- **arch** – The target architecture (auto-detected otherwise).
- **simos** – a SimOS class to use for this project.
- **engine** – The SimEngine class to use for this project.
- **translation_cache** (*bool*) – If True, cache translated basic blocks rather than re-translating them.
- **selfmodifying_code** (bool) – Whether we aggressively support self-modifying code. When enabled, emulation will try to read code from the current state instead of the original memory, regardless of the current memory protections.
- **store_function** – A function that defines how the Project should be stored. Default to pickling.
- **load_function** – A function that defines how the Project should be loaded. Default to unpickling.

- **analyses_preset** (*angr.misc.PluginPreset*) – The plugin preset for the analyses provider (i.e. Analyses instance).

Any additional keyword arguments passed will be passed onto `cle.Loader`.

**Variables**

- **analyses** – The available analyses.
- **entry** – The program entrypoint.
- **factory** – Provides access to important analysis elements such as path groups and symbolic execution results.
- **filename** – The filename of the executable.
- **loader** – The program loader.
- **storage** – Dictionary of things that should be loaded/stored with the Project.

**__init__**(*thing*, *default_analysis_mode=None*, *ignore_functions=None*, *use_sim_procedures=True*, *exclude_sim_procedures_func=None*, *exclude_sim_procedures_list=()*, *arch=None*, *simos=None*, *engine=None*, *load_options=None*, *translation_cache=True*, *selfmodifying_code=False*, *support_selfmodifying_code=None*, *store_function=None*, *load_function=None*, *analyses_preset=None*, *eager_ifunc_resolution=None*, *cache_limits=None*, *rustc_version=None*, *rustc_optimization_level=None*, ***kwargs*)

**Parameters**

- **load_options** (*dict[str, Any] | None*)
- **selfmodifying_code** (*bool*)
- **support_selfmodifying_code** (*bool | None*)
- **cache_limits** (*dict[str, int | None] | None*)

**arch:** *Arch*

**property llm_client**

The LLM client for this project. Lazy-initialized from environment variables on first access. Set manually via `project.llm_client = LLMClient(...)` or configure via environment variables `ANGR_LLM_MODEL`, `ANGR_LLM_API_KEY`, `ANGR_LLM_API_BASE`.

**property kb**

**get_kb**(*name*)

**property analyses:** *AnalysesHubWithDefault*

**hook**(*addr*, *hook=None*, *length=0*, *kwargs=None*, *replace=False*)

Hook a section of code with a custom function. This is used internally to provide symbolic summaries of library functions, and can be used to instrument execution or to modify control flow.

When hook is not specified, it returns a function decorator that allows easy hooking. Usage:

```
# Assuming proj is an instance of angr.Project, we will add a custom hook at␣
↪the entry
# point of the project.
@proj.hook(proj.entry)
def my_hook(state):
    print("Welcome to execution!")
```

**Parameters**

- **addr** – The address to hook.
- **hook** – A `angr.project.Hook` describing a procedure to run at the given address. You may also pass in a SimProcedure class or a function directly and it will be wrapped in a Hook object for you.
- **length** – If you provide a function for the hook, this is the number of bytes that will be skipped by executing the hook by default.
- **kwargs** – If you provide a SimProcedure for the hook, these are the keyword arguments that will be passed to the procedure's *run* method eventually.
- **replace** (`bool` | `None`) – Control the behavior on finding that the address is already hooked. If true, silently replace the hook. If false (default), warn and do not replace the hook. If none, warn and replace the hook.

**is_hooked**(*addr*)

Returns True if *addr* is hooked.

**Parameters**
**addr** – An address.

**Return type**
`bool`

**Returns**
True if addr is hooked, False otherwise.

**hooked_by**(*addr*)

Returns the current hook for *addr*.

**Parameters**
**addr** – An address.

**Return type**
*SimProcedure* | `None`

**Returns**
None if the address is not hooked.

**unhook**(*addr*)

Remove a hook.

**Parameters**
**addr** – The address of the hook.

**hook_symbol**(*symbol_name*, *simproc*, *kwargs=None*, *replace=None*)

Resolve a dependency in a binary. Looks up the address of the given symbol, and then hooks that address. If the symbol was not available in the loaded libraries, this address may be provided by the CLE externs object.

Additionally, if instead of a symbol name you provide an address, some secret functionality will kick in and you will probably just hook that address, UNLESS you're on powerpc64 ABIv1 or some yet-unknown scary ABI that has its function pointers point to something other than the actual functions, in which case it'll do the right thing.

**Parameters**

- **symbol_name** – The name of the dependency to resolve.
- **simproc** – The SimProcedure instance (or function) with which to hook the symbol

- **kwargs** – If you provide a SimProcedure for the hook, these are the keyword arguments that will be passed to the procedure's *run* method eventually.
- **replace** (bool | None) – Control the behavior on finding that the address is already hooked. If true, silently replace the hook. If false, warn and do not replace the hook. If none (default), warn and replace the hook.

**Returns**
The address of the new symbol.

**Return type**
int

**symbol_hooked_by**(*symbol_name*)

Return the SimProcedure, if it exists, for the given symbol name.

**Parameters**
**symbol_name** (*str*) – Name of the symbol.

**Return type**
*SimProcedure* | None

**Returns**
None if the address is not hooked.

**is_symbol_hooked**(*symbol_name*)

Check if a symbol is already hooked.

**Parameters**
**symbol_name** (*str*) – Name of the symbol.

**Returns**
True if the symbol can be resolved and is hooked, False otherwise.

**Return type**
bool

**unhook_symbol**(*symbol_name*)

Remove the hook on a symbol. This function will fail if the symbol is provided by the extern object, as that would result in a state where analysis would be unable to cope with a call to this symbol.

**rehook_symbol**(*new_address*, *symbol_name*, *stubs_on_sync*)

Move the hook for a symbol to a specific address :type new_address: :param new_address: the new address that will trigger the SimProc execution :type symbol_name: :param symbol_name: the name of the symbol (f.i. strcmp ) :return: None

**execute**(**args*, ***kwargs*)

This function is a symbolic execution helper in the simple style supported by triton and manticore. It designed to be run after setting up hooks (see Project.hook), in which the symbolic state can be checked.

This function can be run in three different ways:

- When run with no parameters, this function begins symbolic execution from the entrypoint.
- It can also be run with a "state" parameter specifying a SimState to begin symbolic execution from.
- Finally, it can accept any arbitrary keyword arguments, which are all passed to project.factory.full_init_state.

If symbolic execution finishes, this function returns the resulting simulation manager.

**terminate_execution**()

Terminates a symbolic execution that was started with Project.execute().

**pcode_block_lifter**(*arch*)

Get the p-code basic block lifter, and therefore the Sleigh context, this project decodes *arch* with.

Sleigh records context variables per address, so the blocks of one binary decode with one context and no block of another binary touches it. *arch* is normally this project's own; a *Block* may name another, and decoding that with this project's context would answer for the wrong architecture, so it gets its own.

**Return type**
*PcodeBasicBlockLifter*

**Parameters**
**arch** (*Arch*)

**languages**()

**Return type**
list[str]

**property language_confidence: str | None**

How sure LanguageDetector is about `languages()` ("high", "medium" or "low"), or None when the language did not come from detection.

**property language_is_certain: bool**

Whether the detected language is solid enough to base language-specific ABI decisions on. A single matching symbol name is enough for LanguageDetector to report a language, but not enough to reinterpret every argument and return value in the binary.

**property is_rust_binary: bool**

**property is_go_binary: bool**

Whether the main binary was confidently identified as Go. Go's ABI differs from the platform default in every argument and return value, so a low-confidence detection is deliberately not enough.

**get_function_cache_limit**()

Get the cache limit for function-level caches.

**Return type**
int | None

**Returns**
The cache limit, or None for disabling the cache.

**get_cfg_node_cache_limit**()

Get the cache limit for CFG node caches.

**Return type**
int | None

**Returns**
The cache limit, or None to disable the cache.

**get_cfg_edge_cache_limit**()

Get the cache limit for CFG edge caches (adjacency data spilling).

**Return type**
int | None

**Returns**
The cache limit, or None to disable the cache.

**class** angr.**Server**

Bases: object

Server implements the analysis server with a series of control interfaces exposed.

**Variables**

- **project** – An instance of angr.Project.
- **spill_yard** (*str*) – A directory to store spilled states.
- **db** (*str*) – Path of the database that stores information about spilled states.
- **max_workers** (*int*) – Maximum number of workers. Each worker starts a new process.
- **max_states** (*int*) – Maximum number of active states for each worker.
- **staging_max** (*int*) – Maximum number of inactive states that are kept into memory before spilled onto the disk and potentially be picked up by another worker.
- **bucketizer** (*bool*) – Use the Bucketizer exploration strategy.
- **_worker_exit_callback** – A method that will be called upon the exit of each worker.

**__init__**(*project*, *spill_yard=None*, *db=None*, *max_workers=None*, *max_states=10*, *staging_max=10*, *bucketizer=True*, *recursion_limit=1000*, *worker_exit_callback=None*, *techniques=None*, *add_options=None*, *remove_options=None*)

**inc_active_workers**()

**dec_active_workers**()

**stop**()

**property active_workers**

**property stopped**

**on_worker_exit**(*worker_id*, *stashes*)

**run**()

**exception** angr.**SimAbstractMemoryError**

Bases: *SimMemoryError*

**exception** angr.**SimActionError**

Bases: *SimError*

**class** angr.**SimCC**

Bases: object

A calling convention allows you to extract from a state the data passed from function to function by calls and returns. Most of the methods provided by SimCC that operate on a state assume that the program is just after a call but just before stack frame allocation, though this may be overridden with the *stack_base* parameter to each individual method.

This is the base class for all calling conventions.

__init__(*arch*)

**Parameters**
**arch** (Arch) – The Archinfo arch for this CC

ARG_REGS: list[str] = []

FP_ARG_REGS: list[str] = []

STACKARG_SP_BUFF = 0

STACKARG_SP_DIFF = 0

CALLER_SAVED_REGS: list[str] = []

RETURN_ADDR: *SimFunctionArgument* | None = None

RETURN_VAL: *SimFunctionArgument* | None = None

OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = None

FP_RETURN_VAL: *SimFunctionArgument* | None = None

ARCH: *type*[Arch] | None = None

EXTRA_ARCHES: tuple[*type*[Arch], ...] = ()

CALLEE_CLEANUP = False

STACK_ALIGNMENT = 1

ARG_SLOT_SIZE: int | None = None

ARG_REG_SANITY_FILTER = False

STRICT_CALLER_SAVED_MATCH = True

classmethod arg_reg_offsets(*arch*)

Every register file offset covered by this convention's argument registers.

**Return type**
set[int]

**Parameters**
**arch** (*Arch*)

property arg_slot_size: int

The width in bytes of one argument slot. See ARG_SLOT_SIZE.

property int_args

Iterate through all the possible arg positions that can only be used to store integer or pointer values.

Returns an iterator of SimFunctionArguments

property memory_args

Iterate through all the possible arg positions that can be used to store any kind of argument.

Returns an iterator of SimFunctionArguments

property fp_args

Iterate through all the possible arg positions that can only be used to store floating point values.

Returns an iterator of SimFunctionArguments

is_fp_arg(*arg*)

This should take a SimFunctionArgument instance and return whether or not that argument is a floating-point argument.

**Returns True for MUST be a floating point arg,**
False for MUST NOT be a floating point arg, None for when it can be either.

class ArgSession

Bases: object

A class to keep track of the state accumulated in laying parameters out into memory

both_iter

cc

fp_iter

int_iter

__init__(*cc*)

getstate()

setstate(*state*)

arg_session(*ret_ty*)

Return an arg session.

A session provides the control interface necessary to describe how integral and floating-point arguments are laid out into memory. The default behavior is that there are a finite list of int-only and fp-only argument slots, and an infinite number of generic slots, and when an argument of a given type is requested, the most slot available is used. If you need different behavior, subclass ArgSession.

You need to provide the return type of the function in order to kick off an arg layout session.

**Return type**
*ArgSession*

**Parameters**
**ret_ty** (SimType | *None*)

return_in_implicit_outparam(*ty*)

**Return type**
bool

stack_space(*args*)

**Parameters**
**args** – A list of SimFunctionArguments

**Returns**
The number of bytes that should be allocated on the stack to store all these args, NOT INCLUDING the return address.

**return_val**(*ty*, *perspective_returned=False*)

The location the return value is stored, based on its type.

**property return_addr**

The location the return address is stored.

**next_arg**(*session*, *arg_type*)

**Return type**
*SimFunctionArgument*

**Parameters**

- **session** (ArgSession)
- **arg_type** (SimType)

**static is_fp_value**(*val*)

**static guess_prototype**(*args*, *prototype=None*)

Come up with a plausible SimTypeFunction for the given args (as would be passed to e.g. setup_callsite).

You can pass a variadic function prototype in the *base_type* parameter and all its arguments will be used, only guessing types for the variadic arguments.

**arg_locs**(*prototype*)

**Return type**
list[*SimFunctionArgument*]

**get_args**(*state*, *prototype*, *stack_base=None*)

**set_return_val**(*state*, *val*, *ty*, *stack_base=None*, *perspective_returned=False*)

**setup_callsite**(*state*, *ret_addr*, *args*, *prototype*, *stack_base=None*, *alloc_base=None*, *grow_like_stack=True*)

This function performs the actions of the caller getting ready to jump into a function.

**Parameters**

- **state** – The SimState to operate on
- **ret_addr** – The address to return to when the called function finishes
- **args** – The list of arguments that that the called function will see
- **prototype** – The signature of the call you're making. Should include variadic args concretely.
- **stack_base** – An optional pointer to use as the top of the stack, circa the function entry point
- **alloc_base** – An optional pointer to use as the place to put excess argument data
- **grow_like_stack** – When allocating data at alloc_base, whether to allocate at decreasing addresses

The idea here is that you can provide almost any kind of python type in *args* and it'll be translated to a binary format to be placed into simulated memory. Lists (representing arrays) must be entirely elements of the same type and size, while tuples (representing structs) can be elements of any type and size. If you'd like there to be a pointer to a given value, wrap the value in a *PointerWrapper*.

If stack_base is not provided, the current stack pointer will be used, and it will be updated. If alloc_base is not provided, the stack base will be used and grow_like_stack will implicitly be True.

grow_like_stack controls the behavior of allocating data at alloc_base. When data from args needs to be wrapped in a pointer, the pointer needs to point somewhere, so that data is dumped into memory at alloc_base. If you set alloc_base to point to somewhere other than the stack, set grow_like_stack to False so that sequential allocations happen at increasing addresses.

**teardown_callsite**(*state*, *return_val=None*, *prototype=None*, *force_callee_cleanup=False*)

This function performs the actions of the callee as it's getting ready to return. It returns the address to return to.

**Parameters**

- **state** – The state to mutate
- **return_val** – The value to return
- **prototype** – The prototype of the given function
- **force_callee_cleanup** – If we should clean up the stack allocation for the arguments even if it's not the callee's job to do so

TODO: support the stack_base parameter from setup_callsite… ? Does that make sense in this context? Maybe it could make sense by saying that you pass it in as something like the "saved base pointer" value?

**static find_cc**(*arch*, *args*, *sp_delta*, *platform='Linux'*, *unused_hint=None*, *extra_pop=None*, *language=None*)

Pinpoint the best-fit calling convention and return the corresponding SimCC instance, or None if no fit is found.

**Parameters**

- **arch** (Arch) – An ArchX instance. Can be obtained from archinfo.
- **args** (list[*SimRegArg* | *SimStackArg*]) – A list of arguments. It may be updated by the first matched calling convention to remove non-argument arguments.
- **sp_delta** (int) – The change of stack pointer before and after the call is made.
- **extra_pop** (int | None) – The number of bytes that are popped by the callee. This is used to distinguish between callee-cleanup and caller-cleanup conventions.
- **language** (str | None) – The source language of the binary (e.g. "go"), if known. Languages with their own ABI are matched against that ABI alone.
- **platform** (*str | None*)
- **unused_hint** (*list[SimRegArg] | None*)

**Return type**
*SimCC* | None

**Returns**
A calling convention instance, or None if none of the SimCC subclasses seems to fit the arguments provided.

**classmethod arches**()

**Return type**
tuple[type[Arch], ...]

**get_arg_info**(*state*, *prototype*)

This is just a simple wrapper that collects the information from various locations prototype is as passed to self.arg_locs and self.get_args :type angr.SimState state: :param angr.SimState state: The state to evaluate and extract the values from :return: A list of tuples, where the nth tuple is (type, name, location, value) of the nth argument

**exception** angr.**SimCCError**

Bases: *SimError*

**exception** angr.**SimCCallError**

Bases: *SimExpressionError*

**exception** angr.**SimConcreteBreakpointError**

Bases: *AngrError*

**exception** angr.**SimConcreteMemoryError**

Bases: *AngrError*

**exception** angr.**SimConcreteRegisterError**

Bases: *AngrError*

**exception** angr.**SimEmptyCallStackError**

Bases: *SimError*

**exception** angr.**SimEngineError**

Bases: *SimError*

**exception** angr.**SimError**

Bases: Exception

**bbl_addr = None**

**stmt_idx = None**

**ins_addr = None**

**executed_instruction_count = None**

**guard = None**

**record_state**(*state*)

**exception** angr.**SimEventError**

Bases: *SimStateError*

**exception** angr.**SimException**

Bases: *SimError*

**exception** angr.**SimExpressionError**

Bases: *SimError*

**exception** angr.**SimFastMemoryError**

Bases: *SimMemoryError*

**exception** angr.**SimFastPathError**

Bases: *SimEngineError*

**class** angr.**SimFile**

Bases: *SimFileBase*, *DefaultMemory*

The normal SimFile is meant to model files on disk. It subclasses SimSymbolicMemory so loads and stores to/from it are very simple.

**Parameters**

- **name** – The name of the file

- **content** – Optional initial content for the file as a string or bitvector
- **size** – Optional size of the file. If content is not specified, it defaults to zero
- **has_end** – Whether the size boundary is treated as the end of the file or a frontier at which new content will be generated. If unspecified, will pick its value based on options.FILES_HAVE_EOF. Another caveat is that if the size is also unspecified this value will default to False.
- **seekable** – Optional bool indicating whether seek operations on this file should succeed, default True.
- **writable** – Whether writing to this file is allowed
- **concrete** – Whether or not this file contains mostly concrete data. Will be used by some SimProcedures to choose how to handle variable-length operations like fgets.

**Variables**

**has_end** – Whether this file has an EOF

**__init__**(*name=None*, *content=None*, *size=None*, *has_end=None*, *seekable=True*, *writable=True*, *ident=None*, *concrete=None*, ***kwargs*)

**property category**

reg, mem, or file.

**Type**

Return the category of this SimMemory instance. It can be one of the three following categories

**property size**

The number of data bytes stored by the file at present. May be a symbolic value.

**concretize**(***kwargs*)

Return a concretization of the contents of the file, as a flat bytestring.

**class** angr.**SimFileBase**

Bases: *SimStatePlugin*

SimFiles are the storage mechanisms used by SimFileDescriptors.

Different types of SimFiles can have drastically different interfaces, and as a result there's not much that can be specified on this base class. All the read and write methods take a `pos` argument, which may have different semantics per-class. `0` will always be a valid position to use, though, and the next position you should use is part of the return tuple.

Some simfiles are "streams", meaning that the position that reads come from is determined not by the position you pass in (it will in fact be ignored), but by an internal variable. This is stored as `.pos` if you care to read it. Don't write to it. The same lack-of-semantics applies to this field as well.

**Variables**

- **name** – The name of the file. Purely for cosmetic purposes
- **ident** – The identifier of the file, typically autogenerated from the name and a nonce. Purely for cosmetic purposes, but does appear in symbolic values autogenerated in the file.
- **seekable** – Bool indicating whether seek operations on this file should succeed. If this is True, then `pos` must be a number of bytes from the start of the file.
- **writable** – Bool indicating whether writing to this file is allowed.
- **pos** – If the file is a stream, this will be the current position. Otherwise, None.

- **concrete** – Whether or not this file contains mostly concrete data. Will be used by some SimProcedures to choose how to handle variable-length operations like fgets.
- **file_exists** – Set to False, if file does not exists, set to a claripy Bool if unknown, default True.

**seekable = False**

**pos = None**

**__init__**(*name=None*, *writable=True*, *ident=None*, *concrete=False*, *file_exists=True*, ***kwargs*)

**static make_ident**(*name*)

**concretize**(***kwargs*)

Return a concretization of the contents of the file. The type of the return value of this method will vary depending on which kind of SimFile you're using.

**read**(*pos*, *size*, ***kwargs*)

Read some data from the file.

**Parameters**

- **pos** – The offset in the file to read from.
- **size** – The size to read. May be symbolic.

**Returns**

A tuple of the data read (a bitvector of the length that is the maximum length of the read), the actual size of the read, and the new file position pointer.

**write**(*pos*, *data*, *size=None*, ***kwargs*)

Write some data to the file.

**Parameters**

- **pos** – The offset in the file to write to. May be ignored if the file is a stream or device.
- **data** – The data to write as a bitvector
- **size** – The optional size of the data to write. If not provided will default to the length of the data. Must be constrained to less than or equal to the size of the data.

**Returns**

The new file position pointer.

**property size**

The number of data bytes stored by the file at present. May be a symbolic value.

**class** angr.**SimFileDescriptor**

Bases: *SimFileDescriptorBase*

A simple file descriptor forwarding reads and writes to a SimFile. Contains information about the current opened state of the file, such as the flags or (if relevant) the current position.

**Variables**

- **file** – The SimFile described to by this descriptor
- **flags** – The mode that the file descriptor was opened with, a bitfield of flags

**__init__**(*simfile*, *flags=0*)

**concretize**(***kwargs*)

Return a concretization of the underlying file. Returns whatever format is preferred by the file.

**property file_exists**

This should be True in most cases. Only if we opened an fd of unknown existence, ALL_FILES_EXIST is False and ANY_FILE_MIGHT_EXIST is True, this is a symbolic boolean.

**property read_storage**

Return the SimFile backing reads from this fd

**property write_storage**

Return the SimFile backing writes to this fd

**property read_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**property write_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**class** angr.**SimFileDescriptorDuplex**

Bases: *SimFileDescriptorBase*

A file descriptor that refers to two file storage mechanisms, one to read from and one to write to. As a result, operations like seek, eof, etc no longer make sense.

**Parameters**

- **read_file** – The SimFile to read from
- **write_file** – The SimFile to write to

**__init__**(*read_file*, *write_file*)

**concretize**(***kwargs*)

Return a concretization of the underlying files, as a tuple of (read file, write file).

**property read_storage**

Return the SimFile backing reads from this fd

**property write_storage**

Return the SimFile backing writes to this fd

**property read_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**property write_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**exception** angr.**SimFileError**

Bases: *SimMemoryError*, *SimFilesystemError*

**class** angr.**SimFileStream**

Bases: *SimFile*

A specialized SimFile that uses a flat memory backing, but functions as a stream, tracking its position internally.

The pos argument to the read and write methods will be ignored, and will return None. Instead, there is an attribute `pos` on the file itself, which will give you what you want.

**Parameters**

- **name** – The name of the file, for cosmetic purposes
- **pos** – The initial position of the file, default zero
- **kwargs** – Any other keyword arguments will go on to the SimFile constructor.

**Variables**

**pos** – The current position in the file.

**__init__**(*name=None*, *content=None*, *pos=0*, ***kwargs*)

**exception** angr.**SimFilesystemError**

Bases: *SimError*

**class** angr.**SimHeapBrk**

Bases: *SimHeapBase*

SimHeapBrk represents a trivial heap implementation based on the Unix *brk* system call. This type of heap stores virtually no metadata, so it is up to the user to determine when it is safe to release memory. This also means that it does not properly support standard heap operations like *realloc*.

This heap implementation is a holdover from before any more proper implementations were modelled. At the time, various libc (or win32) SimProcedures handled the heap in the same way that this plugin does now. To make future heap implementations plug-and-playable, they should implement the necessary logic themselves, and dependent SimProcedures should invoke a method by the same name as theirs (prepended with an underscore) upon the heap plugin. Depending on the heap implementation, if the method is not supported, an error should be raised.

Out of consideration for the original way the heap was handled, this plugin implements functionality for all relevant SimProcedures (even those that would not normally be supported together in a single heap implementation).

**Variables**

**heap_location** – the address of the top of the heap, bounding the allocations made starting from *heap_base*

**__init__**(*heap_base=None*, *heap_size=None*)

**allocate**(*sim_size*)

The actual allocation primitive for this heap implementation. Increases the position of the break to allocate space. Has no guards against the heap growing too large.

**Parameters**

**sim_size** – a size specifying how much to increase the break pointer by

**Returns**

a pointer to the previous break position, above which there is now allocated space

release(*sim_size*)

The memory release primitive for this heap implementation. Decreases the position of the break to deallocate space. Guards against releasing beyond the initial heap base.

The mapped extent of the heap is deliberately not shrunk here. Unmapping the released pages would discard their contents, so a release followed by an allocation would hand back blank memory instead of the old bytes, which is a behavior change; and since the extent only ever grows to the high-water mark of the break, keeping it costs nothing beyond what the program already asked for.

**Parameters**

**sim_size** – a size specifying how much to decrease the break pointer by (may be symbolic or not)

exception angr.SimHeapError

Bases: *SimStateError*

class angr.SimHeapPTMalloc

Bases: *SimHeapFreelist*

A freelist-style heap implementation inspired by ptmalloc. The chunks used by this heap contain heap metadata in addition to user data. While the real-world ptmalloc is implemented using multiple lists of free chunks (corresponding to their different sizes), this more basic model uses a single list of chunks and searches for free chunks using a first-fit algorithm.

**NOTE:** The plugin must be registered using `register_plugin` with name `heap` in order to function properly.

**Variables**

- **heap_base** – the address of the base of the heap in memory
- **heap_size** – the total size of the main memory region managed by the heap in memory
- **mmap_base** – the address of the region from which large mmap allocations will be made
- **free_head_chunk** – the head of the linked list of free chunks in the heap

__init__(*heap_base=None*, *heap_size=None*)

chunk_from_mem(*ptr*)

Given a pointer to a user payload, return the base of the chunk associated with that payload (i.e. the chunk pointer). Returns None if ptr is null.

**Parameters**

**ptr** – a pointer to the base of a user payload in the heap

**Returns**

a pointer to the base of the associated heap chunk, or None if ptr is null

class angr.SimHostFilesystem

Bases: *SimConcreteFilesystem*

Simulated mount that makes some piece from the host filesystem available to the guest.

**Parameters**

- **host_path** (*str*) – The path on the host to mount
- **pathsep** (*str*) – The host path separator character, default os.path.sep

__init__(*host_path=None*, ***kwargs*)

exception angr.SimIRSBError

Bases: *SimEngineError*

**exception** angr.**SimIRSBNoDecodeError**

Bases: *SimIRSBError*

**exception** angr.**SimMemoryAddressError**

Bases: *SimMemoryError*

**exception** angr.**SimMemoryError**

Bases: *SimStateError*

**exception** angr.**SimMemoryLimitError**

Bases: *SimMemoryError*

**exception** angr.**SimMemoryMissingError**

Bases: *SimMemoryError*

**exception** angr.**SimMergeError**

Bases: *SimStateError*

**exception** angr.**SimMissingTempError**

Bases: *SimValueError*, IndexError

**class** angr.**SimMount**

Bases: *SimStatePlugin*

This is the base class for "mount points" in angr's simulated filesystem. Subclass this class and give it to the filesystem to intercept all file creations and opens below the mountpoint. Since this a SimStatePlugin you may also want to implement set_state, copy, merge, etc.

**get**(*path_elements*)

Implement this function to instrument file lookups.

**Parameters**
**path_elements** – A list of path elements traversing from the mountpoint to the file

**Returns**
A SimFile, or None

**insert**(*path_elements*, *simfile*)

Implement this function to instrument file creation.

**Parameters**

- **path_elements** – A list of path elements traversing from the mountpoint to the file
- **simfile** – The file to insert

**Returns**
A bool indicating whether the insert occurred

**delete**(*path_elements*)

Implement this function to instrument file deletion.

**Parameters**
**path_elements** – A list of path elements traversing from the mountpoint to the file

**Returns**
A bool indicating whether the delete occurred

**lookup**(*sim_file*)

Look up the path of a SimFile in the mountpoint

**Parameters**
**sim_file** – A SimFile object needs to be looked up

**Returns**
A string representing the path of the file in the mountpoint Or None if the SimFile does not exist in the mountpoint

class angr.SimOS

Bases: object

A class describing OS/arch-level configuration.

__init__(*project*, *name=None*)

**Parameters**
- **project** (Project)
- **name** (*str | None*)

name: str | None

configure_project()
Configure the project to set up global settings (like SimProcedures).

state_blank(*addr=None*, *initial_prefix=None*, *brk=None*, *stack_end=None*, *stack_size=8388608*, *stdin=None*, *thread_idx=None*, *permissions_backer=None*, ***kwargs*)
Initialize a blank state.

All parameters are optional.

**Parameters**
- **addr** – The execution start address.
- **initial_prefix**
- **stack_end** – The end of the stack (i.e., the byte after the last valid stack address).
- **stack_size** – The number of bytes to allocate for stack space
- **brk** – The address of the process' break.

**Returns**
The initialized SimState.

Any additional arguments will be passed to the SimState constructor

state_entry(***kwargs*)

state_full_init(***kwargs*)

state_call(*addr*, **args*, ***kwargs*)

prepare_call_state(*calling_state*, *initial_state=None*, *preserve_registers=()*, *preserve_memory=()*)
This function prepares a state that is executing a call instruction. If given an initial_state, it copies over all of the critical registers to it from the calling_state. Otherwise, it prepares the calling_state for action.

This is mostly used to create minimalistic for CFG generation. Some ABIs, such as MIPS PIE and x86 PIE, require certain information to be maintained in certain registers. For example, for PIE MIPS, this function transfer t9, gp, and ra to the new state.

**prepare_function_symbol**(*symbol_name*, *basic_addr=None*)

Prepare the address space with the data necessary to perform relocations pointing to the given symbol

Returns a 2-tuple. The first item is the address of the function code, the second is the address of the relocation target.

**handle_exception**(*successors*, *engine*, *exception*)

Perform exception handling. This method will be called when, during execution, a SimException is thrown. Currently, this can only indicate a segfault, but in the future it could indicate any unexpected exceptional behavior that can't be handled by ordinary control flow.

The method may mutate the provided SimSuccessors object in any way it likes, or re-raise the exception.

**Parameters**

- **successors** – The SimSuccessors object currently being executed on
- **engine** – The engine that was processing this step
- **exception** – The actual exception object

**syscall**(*state*, *allow_unsupported=True*)

**Return type**
*SimProcedure* | None

**Parameters**

- **state** (SimState)
- **allow_unsupported** (*bool*)

**syscall_abi**(*state*)

**Return type**
str | None

**Parameters**
**state** (SimState)

**syscall_cc**(*state*)

**Return type**
*SimCCSyscall* | None

**Parameters**
**state** (SimState)

**is_syscall_addr**(*addr*)

**Return type**
bool

**syscall_from_addr**(*addr*, *allow_unsupported=True*)

**Return type**
*SimProcedure* | None

**syscall_from_number**(*number*, *allow_unsupported=True*, *abi=None*)

**Return type**
*SimProcedure* | None

setup_gdt(*state*, *gdt*)

Write the GlobalDescriptorTable object in the current state memory

**Parameters**

- **state** – state in which to write the GDT
- **gdt** – GlobalDescriptorTable object

**Returns**

generate_gdt(*fs*, *gs*, *fs_size=4294967295*, *gs_size=4294967295*)

Generate a GlobalDescriptorTable object and populate it using the value of the gs and fs register

**Parameters**

- **fs** – value of the fs segment register
- **gs** – value of the gs segment register
- **fs_size** – size of the fs segment register
- **gs_size** – size of the gs segment register

**Returns**

gdt a GlobalDescriptorTable object

exception angr.SimOperationError

Bases: *SimError*

class angr.SimPackets

Bases: *SimFileBase*

The SimPackets is meant to model inputs whose content is delivered a series of asynchronous chunks. The data is stored as a list of read or write results. For symbolic sizes, state.libc.max_packet_size will be respected. If the SHORT_READS option is enabled, reads will return a symbolic size constrained to be less than or equal to the requested size.

A SimPackets cannot be used for both reading and writing - for socket objects that can be both read and written to you should use a file descriptor to multiplex the read and write operations into two separate file storage mechanisms.

**Parameters**

- **name** – The name of the file, for cosmetic purposes
- **write_mode** – Whether this file is opened in read or write mode. If this is unspecified it will be autodetected.
- **content** – Some initial content to use for the file. Can be a list of bytestrings or a list of tuples of content ASTs and size ASTs.

**Variables**

- **write_mode** – See the eponymous parameter
- **content** – A list of packets, as tuples of content ASTs and size ASTs.

__init__(*name*, *write_mode=None*, *content=None*, *writable=True*, *ident=None*, ***kwargs*)

property size

The number of data bytes stored by the file at present. May be a symbolic value.

**concretize**(***kwargs*)

Returns a list of the packets read or written as bytestrings.

**read**(*pos*, *size*, ***kwargs*)

Read a packet from the stream.

**Parameters**

- **pos** (*int*) – The packet number to read from the sequence of the stream. May be None to append to the stream.
- **size** – The size to read. May be symbolic.
- **short_reads** – Whether to replace the size with a symbolic value constrained to less than or equal to the original size. If unspecified, will be chosen based on the state option.

**Returns**

A tuple of the data read (a bitvector of the length that is the maximum length of the read) and the actual size of the read.

**write**(*pos*, *data*, *size=None*, *events=True*, ***kwargs*)

Write a packet to the stream.

**Parameters**

- **pos** (*int*) – The packet number to write in the sequence of the stream. May be None to append to the stream.
- **data** – The data to write, as a string or bitvector.
- **size** – The optional size to write. May be symbolic; must be constrained to at most the size of data.

**Returns**

The next packet to use after this

**class** angr.**SimPacketsStream**

Bases: *SimPackets*

A specialized SimPackets that tracks its position internally.

The pos argument to the read and write methods will be ignored, and will return None. Instead, there is an attribute `pos` on the file itself, which will give you what you want.

**Parameters**

- **name** – The name of the file, for cosmetic purposes
- **pos** – The initial position of the file, default zero
- **kwargs** – Any other keyword arguments will go on to the SimPackets constructor.

**Variables**

**pos** – The current position in the file.

**__init__**(*name*, *pos=0*, ***kwargs*)

**exception** angr.**SimPosixError**

Bases: *SimStateError*

**class** angr.**SimProcedure**

Bases: `object`

A SimProcedure is a wonderful object which describes a procedure to run on a state.

You may subclass SimProcedure and override `run()`, replacing it with mutating `self.state` however you like, and then either returning a value or jumping away somehow.

A detailed discussion of programming SimProcedures may be found at https://docs.angr.io/extending-angr/simprocedures

**Parameters**

- **arch** – The architecture to use for this procedure
- **symbolic_return** – Whether the procedure's return value should be stubbed into a single symbolic variable constratined to the real return value
- **returns** – Whether the procedure should return to its caller afterwards
- **is_syscall** – Whether this procedure is a syscall
- **num_args** – The number of arguments this procedure should extract
- **display_name** – The name to use when displaying this procedure
- **library_name** – The name of the library from which the function we're emulating comes
- **cc** (*SimCC* | None) – The SimCC to use for this procedure
- **sim_kwargs** – Additional keyword arguments to be passed to run()
- **is_function** – Whether this procedure emulates a function

The following class variables should be set if necessary when implementing a new SimProcedure:

**Variables**

- **NO_RET** – Set this to true if control flow will never return from this function
- **DYNAMIC_RET** – Set this to true if whether the control flow returns from this function or not depends on the context (e.g., libc's error() call). Must implement dynamic_returns() method.
- **ADDS_EXITS** – Set this to true if you do any control flow other than returning
- **IS_FUNCTION** – Does this procedure simulate a function? True by default
- **ARGS_MISMATCH** – Does this procedure have a different list of arguments than what is provided in the function specification? This may happen when we manually extract arguments in the run() method of a SimProcedure. False by default.
- **local_vars** – If you use `self.call()`, set this to a list of all the local variable names in your class. They will be restored on return.

The following instance variables are available when working with simprocedures from the inside or the outside:

**Variables**

- **project** – The associated angr project
- **arch** – The associated architecture
- **addr** – The linear address at which the procedure is executing
- **cc** – The calling convention in use for engaging with the ABI
- **canonical** – The canonical version of this SimProcedure. Procedures are deepcopied for many reasons, including to be able to store state related to a specific run and to be able to hook continuations.
- **kwargs** – Any extra keyword arguments used to construct the procedure; will be passed to `run`

- **display_name** – See the eponymous parameter
- **library_name** – See the eponymous parameter
- **abi** – If this is a syscall simprocedure, which ABI are we using to map the syscall numbers?
- **symbolic_return** – See the eponymous parameter
- **syscall_number** – If this procedure is a syscall, the number will be populated here.
- **returns** – See eponymous parameter and NO_RET cvar
- **is_syscall** – See eponymous parameter
- **is_function** – See eponymous parameter and cvar
- **is_stub** – See eponymous parameter
- **is_continuation** – Whether this procedure is the original or a continuation resulting from `self.call()`
- **continuations** – A mapping from name to each known continuation
- **run_func** – The name of the function implementing the procedure. "run" by default, but different in continuations.
- **num_args** – The number of arguments to the procedure. If not provided in the parameter, extracted from the definition of `self.run`

The following instance variables are only used in a copy of the procedure that is actually executing on a state:

**Variables**

- **state** – The SimState we should be mutating to perform the procedure
- **successors** – The SimSuccessors associated with the current step
- **arguments** – The function arguments, deserialized from the state
- **arg_session** (*ArgSession* | `int` | `None`) – The ArgSession that was used to parse arguments out of the state, in case you need it for varargs
- **use_state_arguments** – Whether we're using arguments extracted from the state or manually provided
- **ret_to** – The current return address
- **ret_expr** – The computed return value
- **call_ret_expr** – The return value from having used `self.call()`
- **inhibit_autoret** – Whether we should avoid automatically adding an exit for returning once the run function ends
- **arg_session** – The ArgSession object that was used to extract the runtime argument values. Useful for if you want to extract variadic args.

**__init__**(*project=None*, *cc=None*, *prototype=None*, *symbolic_return=None*, *returns=None*, *is_syscall=False*, *is_stub=False*, *num_args=None*, *display_name=None*, *library_name=None*, *is_function=None*, ***kwargs*)

**Parameters**

- **project** (`Project` | *None*)
- **cc** (`SimCC` | *None*)
- **prototype** (*str* | `SimTypeFunction` | *None*)

**state:** *SimState*

**execute**(*state*, *successors=None*, *arguments=None*, *ret_to=None*)

Call this method with a SimState and a SimSuccessors to execute the procedure.

Alternately, successors may be none if this is an inline call. In that case, you should provide arguments to the function.

**make_continuation**(*name*)

**NO_RET = False**

**DYNAMIC_RET = False**

**ADDS_EXITS = False**

**IS_FUNCTION = True**

**ARGS_MISMATCH = False**

**ALT_NAMES = None**

**local_vars:** tuple[str, ...] = ()

**run**(**args*, ***kwargs*)

Implement the actual procedure here!

**Return type**
Any

**static_exits**(*blocks*, ***kwargs*)

Get new exits by performing static analysis and heuristics. This is a fast and best-effort approach to get new exits for scenarios where states are not available (e.g. when building a fast CFG).

**Parameters**
**blocks** (*list*) – Blocks that are executed before reaching this SimProcedure.

**Returns**
A list of dicts. Each dict should contain the following entries: 'address', 'jumpkind', and 'namehint'.

**Return type**
list

**dynamic_returns**(*blocks*, ***kwargs*)

Determines if a call to this function returns or not by performing static analysis and heuristics.

**Parameters**
**blocks** – Blocks that are executed before reaching this SimProcedure.

**Return type**
bool

**Returns**
True if the call returns, False otherwise.

**property should_add_successors**

**set_args**(*args*)

**va_arg**(*ty*, *index=None*)

**inline_call**(*procedure*, **arguments*, ***kwargs*)

Call another SimProcedure in-line to retrieve its return value. Returns an instance of the procedure with the ret_expr property set.

**Parameters**

- **procedure** – The class of the procedure to execute
- **arguments** – Any additional positional args will be used as arguments to the procedure call
- **sim_kwargs** – Any additional keyword args will be passed as sim_kwargs to the procedure constructor

**fix_prototype_returnty**(*ret_size*)

**ret**(*expr=None*)

Add an exit representing a return from this function. If this is not an inline call, grab a return address from the state and jump to it. If this is not an inline call, set a return expression with the calling convention.

**call**(*addr*, *args*, *continue_at*, *cc=None*, *prototype=None*, *jumpkind='Ijk_Call'*)

Add an exit representing calling another function via pointer.

**Parameters**

- **addr** – The address of the function to call
- **args** – The list of arguments to call the function with
- **continue_at** – Later, when the called function returns, execution of the current procedure will continue in the named method.
- **cc** – Optional: use this calling convention for calling the new function. Default is to use the current convention.
- **prototype** – Optional: The prototype to use for the call. Will default to all-ints.

**jump**(*addr*, *jumpkind='Ijk_Boring'*)

Add an exit representing jumping to an address.

**exit**(*exit_code*)

Add an exit representing terminating the program.

**property is_java**

**property argument_types**

**property return_type**

**exception** angr.**SimProcedureArgumentError**

Bases: *SimProcedureError*

**exception** angr.**SimProcedureError**

Bases: *SimEngineError*

**exception** angr.**SimRegionMapError**

Bases: *SimMemoryError*

**exception** angr.**SimReliftException**

Bases: *SimEngineError*

angr.**SimSegfaultError**

alias of *SimSegfaultException*

**exception** angr.**SimSegfaultException**

Bases: *SimException*, *SimMemoryError*

**exception** angr.**SimShadowStackError**

Bases: *SimProcedureError*

**exception** angr.**SimSlicerError**

Bases: *SimError*

**exception** angr.**SimSolverError**

Bases: *SimError*

**exception** angr.**SimSolverModeError**

Bases: *SimSolverError*

**exception** angr.**SimSolverOptionError**

Bases: *SimSolverError*

**class** angr.**SimState**

Bases: *PluginHub*[*SimStatePlugin*], Generic

The SimState represents the state of a program, including its memory, registers, and so forth.

**Parameters**

**project** (*Project*) – The project instance.

**Variables**

- **regs** – A convenient view of the state's registers, where each register is a property
- **mem** – A convenient view of the state's memory, a *angr.state_plugins.view.SimMemView*
- **registers** – The state's register file as a flat memory region
- **memory** – The state's memory as a flat memory region
- **solver** – The symbolic solver and variable manager for this state
- **inspect** – The breakpoint manager, a *angr.state_plugins.inspect.SimInspector*
- **log** – Information about the state's history
- **scratch** – Information about the current execution step
- **posix** – MISNOMER: information about the operating system or environment model
- **fs** – The current state of the simulated filesystem
- **libc** – Information about the standard library we are emulating
- **cgc** – Information about the cgc environment
- **uc_manager** – Control of under-constrained symbolic execution
- **unicorn** – Control of the Unicorn Engine

**solver:** *SimSolver*

**posix:** *SimSystemPosix*

**fs:** *SimFilesystem*

**libc:** *SimStateLibc*

**registers:** *DefaultMemory*

**regs:** *SimRegNameView*

**memory:** *DefaultMemory*

**callstack:** *CallStack*

**mem:** *SimMemView*

**history:** *SimStateHistory*

**inspect:** *SimInspector*

**jni_references:** *SimStateJNIReferences*

**scratch:** *SimStateScratch*

**heap:** *SimHeapBase*

**__init__**(*project*, *plugins=None*, *mode=None*, *options=None*, *add_options=None*, *remove_options=None*, *special_memory_filler=None*, *os_name=None*, *plugin_preset='default'*, *cle_memory_backer=None*, *dict_memory_backer=None*, *permissions_map=None*, *default_permissions=3*, *stack_perms=None*, *stack_end=None*, *stack_size=None*, *regioned_memory_cls=None*, ***kwargs*)

**Parameters**

- **project** (Project)
- **plugins** (*dict[str, SimStatePlugin] | None*)
- **mode** (*str | None*)
- **options** (*set[str] | list[str] | SimStateOptions | None*)
- **add_options** (*set[str] | None*)
- **remove_options** (*set[str] | None*)
- **special_memory_filler** (*Callable[[str, int, int, SimState], Any] | None*)
- **os_name** (*str | None*)
- **plugin_preset** (*str*)
- **cle_memory_backer** (*Clemory | None*)
- **dict_memory_backer** (*dict[int, bytes] | None*)
- **permissions_map** (*dict[tuple[int, int], int] | None*)
- **default_permissions** (*int*)
- **stack_perms** (*int | None*)
- **stack_end** (*int | None*)
- **stack_size** (*int | None*)

**property plugins**

**property ip**

Get the instruction pointer expression, trigger SimInspect breakpoints, and generate SimActions. Use `_ip` to not trigger breakpoints or generate actions.

**Returns**
an expression

**property addr: IPTypeConc**

Get the concrete address of the instruction pointer, without triggering SimInspect breakpoints or generating SimActions. An integer is returned, or an exception is raised if the instruction pointer is symbolic.

**Returns**
an int

**property arch: Arch**

The architecture of the state. This is the project's architecture, except for JavaVM projects with JNI support, where it is either the Soot or the native architecture, depending on whether the current ip is a Soot address.

**property javavm_memory**

In case of an JavaVM with JNI support, a state can store the memory plugin twice; one for the native and one for the java view of the state.

**Returns**
The JavaVM view of the memory plugin.

**property javavm_registers**

In case of an JavaVM with JNI support, a state can store the registers plugin twice; one for the native and one for the java view of the state.

**Returns**
The JavaVM view of the registers plugin.

**simplify**(**args*)

Simplify this state's constraints.

**add_constraints**(**constraints*)

Add some constraints to the state.

You may pass in any number of symbolic booleans as variadic positional arguments.

**satisfiable**(***kwargs*)

Whether the state's constraints are satisfiable

**downsize**()

Clean up after the solver engine. Calling this when a state no longer needs to be solved on will reduce memory usage.

**step**(***kwargs*)

Perform a step of symbolic execution using this state. Any arguments to *AngrObjectFactory.successors* can be passed to this.

**Returns**
A SimSuccessors object categorizing the results of the step.

**block**(**args*, ***kwargs*)

Represent the basic block at this state's instruction pointer. Any arguments to *AngrObjectFactory.block* can ba passed to this.

**Returns**
A Block object describing the basic block of code at this point.

`copy()`

Returns a copy of the state.

`merge(*others, **kwargs)`

Merges this state with the other states. Returns the merging result, merged state, and the merge flag.

**Parameters**

- **states** – the states to merge
- **merge_conditions** – a tuple of the conditions under which each state holds
- **common_ancestor** – a state that represents the common history between the states being merged. Usually it is only available when EFFICIENT_STATE_MERGING is enabled, otherwise weak-refed states might be dropped from state history instances.
- **plugin_whitelist** – a list of plugin names that will be merged. If this option is given and is not None, any plugin that is not inside this list will not be merged, and will be created as a fresh instance in the new state.
- **common_ancestor_history** – a SimStateHistory instance that represents the common history between the states being merged. This is to allow optimal state merging when EFFICIENT_STATE_MERGING is disabled.

**Returns**
(merged state, merge flag, a bool indicating if any merging occurred)

`reg_concrete(*args, **kwargs)`

Returns the contents of a register but, if that register is symbolic, raises a SimValueError.

`mem_concrete(*args, **kwargs)`

Returns the contents of a memory but, if the contents are symbolic, raises a SimValueError.

`stack_push(thing)`

Push 'thing' to the stack, writing the thing to memory and adjusting the stack pointer.

`stack_pop()`

Pops from the stack and returns the popped thing. The length will be the architecture word size.

`stack_read(offset, length, bp=False)`

Reads length bytes, at an offset into the stack.

**Parameters**

- **offset** – The offset from the stack pointer.
- **length** – The number of bytes to read.
- **bp** – If True, offset from the BP instead of the SP. Default: False.

`make_concrete_int(expr)`

`dbg_print_stack(depth=None, sp=None)`

Only used for debugging purposes. Return the current stack info in formatted string. If depth is None, the current stack frame (from sp to bp) will be printed out.

`set_mode(mode)`

`property thumb`

**exception** angr.**SimStateError**

Bases: *SimError*

**exception** angr.**SimStateOptionsError**

Bases: *SimError*

**class** angr.**SimStatePlugin**

Bases: object

This is a base class for SimState plugins. A SimState plugin will be copied along with the state when the state is branched. They are intended to be used for things such as tracking open files, tracking heap details, and providing storage and persistence for SimProcedures.

**__init__**()

**Return type**
None

**state:** *SimState*[Any, Any]

**set_state**(*state*)

Sets a new state (for example, if the state has been branched)

**Return type**
None

**static memo**(*f*)

A decorator function you should apply to copy

**Return type**
_CopyFunc[TypeVar(S_co)]

**Parameters**
**f** (*Callable[[Any, dict[int, Any]], S_co]*)

**copy**(*_memo*)

Should return a copy of the plugin without any state attached. Should check the memo first, and add itself to memo if it ends up making a new copy.

In order to simplify using the memo, you should annotate implementations of this function with SimStatePlugin.memo

The base implementation of this function constructs a new instance of the plugin's class without calling its initializer. If you super-call down to it, make sure you instantiate all the fields in your copy method!

**Parameters**

- **memo** – A dictionary mapping object identifiers (id(obj)) to their copied instance. Use this to avoid infinite recursion and diverged copies.
- **_memo** (*dict[int, Any]*)

**Return type**
*SimStatePlugin*

**merge**(*others*, *merge_conditions*, *common_ancestor=None*)

Should merge the state plugin with the provided others. This will be called by state.merge() after copying the target state, so this should mutate the current instance to merge with the others.

Note that when multiple instances of a single plugin object (for example, a file) are referenced in the state, it is important that merge only ever be called once. This should be solved by designating one of the plugin's referees as the "real owner", who should be the one to actually merge it. This technique doesn't work to

resolve the similar issue that arises during copying because merging doesn't produce a new reference to insert.

There will be n `others` and n+1 merge conditions, since the first condition corresponds to self. To match elements up to conditions, say `zip([self] + others, merge_conditions)`

When implementing this, make sure that you "deepen" both `others` and `common_ancestor` before calling sub-elements' merge methods, e.g.

```
self.foo.merge(
    [o.foo for o in others],
    merge_conditions,
    common_ancestor=common_ancestor.foo if common_ancestor is not None else None
)
```

During static analysis, merge_conditions can be None, in which case you should use `state.solver.union(values)`. TODO: fish please make this less bullshit

There is a utility `claripy.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = claripy.ite_cases(zip(conditions[1:], [o.bar for o in others]), self.bar)`

**Parameters**

- **others** – the other state plugins to merge with
- **merge_conditions** – a symbolic condition for each of the plugins
- **common_ancestor** – a common ancestor of this plugin and the others being merged

**Returns**
True if the state plugins are actually merged.

**Return type**
bool

**classmethod register_default**(*name*, *xtr=None*)

**Return type**
None

**Parameters**

- **name** (*str*)
- **xtr** (type[SimStatePlugin] | *str* | *None*)

**init_state**()

Use this function to perform any initialization on the state at plugin-add time

**Return type**
None

**exception** angr.**SimStatementError**

Bases: *SimError*

**exception** angr.**SimSymbolicFilesystemError**

Bases: *SimFilesystemError*

**exception** angr.**SimTranslationError**

Bases: *SimEngineError*

**exception** angr.**SimUCManagerAllocationError**

Bases: *SimUCManagerError*

**exception** angr.**SimUCManagerError**

Bases: *SimError*

**exception** angr.**SimUnicornError**

Bases: *SimError*

**exception** angr.**SimUnicornSymbolic**

Bases: *SimError*

**exception** angr.**SimUnicornUnsupport**

Bases: *SimError*

**exception** angr.**SimUninitializedAccessError**

Bases: *SimExpressionError*

**exception** angr.**SimUnsatError**

Bases: *SimValueError*

**exception** angr.**SimUnsupportedError**

Bases: *SimError*

**exception** angr.**SimValueError**

Bases: *SimSolverError*

**exception** angr.**SimZeroDivisionException**

Bases: *SimException*, *SimOperationError*

**class** angr.**SimulationManager**

Bases: `object`

The Simulation Manager is the future future.

Simulation managers allow you to wrangle multiple states in a slick way. States are organized into "stashes", which you can step forward, filter, merge, and move around as you wish. This allows you to, for example, step two different stashes of states at different rates, then merge them together.

Stashes can be accessed as attributes (i.e. .active). A mulpyplexed stash can be retrieved by prepending the name with *mp_*, e.g. *.mp_active*. A single state from the stash can be retrieved by prepending the name with *one_*, e.g. *.one_active*.

Note that you shouldn't usually be constructing SimulationManagers directly - there is a convenient shortcut for creating them in `Project.factory`: see *angr.factory.AngrObjectFactory*.

The most important methods you should look at are `step`, `explore`, and `use_technique`.

**Parameters**

- **project** (`angr.project.Project`) – A Project instance.
- **stashes** – A dictionary to use as the stash store.
- **active_states** – Active states to seed the "active" stash with.
- **hierarchy** – A StateHierarchy object to use to track the relationships between states.
- **resilience** – A set of errors to catch during stepping to put a state in the `errore` list. You may also provide the values False, None (default), or True to catch, respectively, no errors, all angr-specific errors, and a set of many common errors.

- **save_unsat** – Set to True in order to introduce unsatisfiable states into the `unsat` stash instead of discarding them immediately.
- **auto_drop** – A set of stash names which should be treated as garbage chutes.
- **completion_mode** – A function describing how multiple exploration techniques with the `complete` hook set will interact. By default, the builtin function `any`.
- **techniques** – A list of techniques that should be pre-set to use with this manager.
- **suggestions** – Whether to automatically install the Suggestions exploration technique. Default True.

**Variables**

- **errored** – Not a stash, but a list of ErrorRecords. Whenever a step raises an exception that we catch, the state and some information about the error are placed in this list. You can adjust the list of caught exceptions with the *resilience* parameter.
- **stashes** – All the stashes on this instance, as a dictionary.
- **completion_mode** – A function describing how multiple exploration techniques with the `complete` hook set will interact. By default, the builtin function `any`.

**ALL = '_ALL'**

**DROP = '_DROP'**

**__init__**(*project*, *active_states=None*, *stashes=None*, *hierarchy=None*, *resilience=None*, *save_unsat=False*, *auto_drop=None*, *errored=None*, *completion_mode=<built-in function any>*, *techniques=None*, *suggestions=True*, ***kwargs*)

**active:** list[*SimState*]

**stashed:** list[*SimState*]

**pruned:** list[*SimState*]

**unsat:** list[*SimState*]

**deadended:** list[*SimState*]

**unconstrained:** list[*SimState*]

**found:** list[*SimState*]

**one_active:** *SimState*

**one_stashed:** *SimState*

**one_pruned:** *SimState*

**one_unsat:** *SimState*

**one_deadended:** *SimState*

**one_unconstrained:** *SimState*

**one_found:** *SimState*

**property errored:** list[*ErrorRecord*]

property stashes: defaultdict[str, list[SimState]]

mulpyplex(*stashes)

Mulpyplex across several stashes.

**Parameters**
**stashes** – the stashes to mulpyplex

**Returns**
a mulpyplexed list of states from the stashes in question, in the specified order

copy(*deep=False*)

Make a copy of this simulation manager. Pass `deep=True` to copy all the states in it as well.

If the current callstack includes hooked methods, the already-called methods will not be included in the copy.

use_technique(*tech*)

Use an exploration technique with this SimulationManager.

Techniques can be found in *angr.exploration_techniques*.

**Parameters**
**tech** (ExplorationTechnique) – An ExplorationTechnique object that contains code to modify this SimulationManager's behavior.

**Returns**
The technique that was added, for convenience

remove_technique(*tech*)

Remove an exploration technique from a list of active techniques.

**Parameters**
**tech** (ExplorationTechnique) – An ExplorationTechnique object.

explore(*stash='active'*, *n=None*, *find=None*, *avoid=None*, *find_stash='found'*, *avoid_stash='avoid'*, *cfg=None*, *num_find=1*, *avoid_priority=False*, ***kwargs*)

Tick stash "stash" forward (up to "n" times or until "num_find" states are found), looking for condition "find", avoiding condition "avoid". Stores found states into "find_stash' and avoided states into "avoid_stash".

The "find" and "avoid" parameters may be any of:

- An address to find
- A set or list of addresses to find
- A function that takes a state and returns whether or not it matches.

If an angr CFG is passed in as the "cfg" parameter and "find" is either a number or a list or a set, then any states which cannot possibly reach a success state without going through a failure state will be preemptively avoided.

run(*stash='active'*, *n=None*, *until=None*, ***kwargs*)

Run until the SimulationManager has reached a completed state, according to the current exploration techniques. If no exploration techniques that define a completion state are being used, run until there is nothing left to run.

**Parameters**

- **stash** – Operate on this stash
- **n** – Step at most this many times

- **until** – If provided, should be a function that takes a SimulationManager and returns True or False. Stepping will terminate when it is True.

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**complete**()

Returns whether or not this manager has reached a "completed" state.

**step**(*stash='active'*, *target_stash=None*, *n=None*, *selector_func=None*, *step_func=None*, *error_list=None*, *successor_func=None*, *until=None*, *filter_func=None*, ***run_args*)

Step a stash of states forward and categorize the successors appropriately.

The parameters to this function allow you to control everything about the stepping and categorization process.

**Parameters**

- **stash** – The name of the stash to step (default: 'active')
- **target_stash** – The name of the stash to put the results in (default: same as `stash`)
- **error_list** – The list to put ErrorRecord objects in (default: `self.errored`)
- **selector_func** – If provided, should be a function that takes a state and returns a boolean. If True, the state will be stepped. Otherwise, it will be kept as-is.
- **step_func** – If provided, should be a function that takes a SimulationManager and returns a SimulationManager. Will be called with the SimulationManager at every step. Note that this function should not actually perform any stepping - it is meant to be a maintenance function called after each step.
- **successor_func** – If provided, should be a function that takes a state and return its successors. Otherwise, project.factory.successors will be used.
- **filter_func** – If provided, should be a function that takes a state and return the name of the stash, to which the state should be moved.
- **until** – (DEPRECATED) If provided, should be a function that takes a SimulationManager and returns True or False. Stepping will terminate when it is True.
- **n** – (DEPRECATED) The number of times to step (default: 1 if "until" is not provided)

Additionally, you can pass in any of the following keyword args for project.factory.successors:

**Parameters**

- **jumpkind** – The jumpkind of the previous exit
- **addr** – An address to execute at instead of the state's ip.
- **stmt_whitelist** – A list of stmt indexes to which to confine execution.
- **last_stmt** – A statement index at which to stop execution.
- **thumb** – Whether the block should be lifted in ARM's THUMB mode.
- **backup_state** – A state to read bytes from instead of using project memory.
- **opt_level** – The VEX optimization level to use.
- **insn_bytes** – A string of bytes to use for the block instead of the project.

- **size** – The maximum size of the block, in bytes.
- **num_inst** – The maximum number of instructions.
- **traceflags** – traceflags to be passed to VEX. Default: 0

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**step_state**(*state*, *successor_func=None*, *error_list=None*, ***run_args*)

Don't use this function manually - it is meant to interface with exploration techniques.

**filter**(*state*, *filter_func=None*)

Don't use this function manually - it is meant to interface with exploration techniques.

**selector**(*state*, *selector_func=None*)

Don't use this function manually - it is meant to interface with exploration techniques.

**successors**(*state*, *successor_func=None*, ***run_args*)

Don't use this function manually - it is meant to interface with exploration techniques.

**prune**(*filter_func=None*, *from_stash='active'*, *to_stash='pruned'*)

Prune unsatisfiable states from a stash.

This function will move all unsatisfiable states in the given stash into a different stash.

**Parameters**

- **filter_func** – Only prune states that match this filter.
- **from_stash** – Prune states from this stash. (default: 'active')
- **to_stash** – Put pruned states in this stash. (default: 'pruned')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**populate**(*stash*, *states*)

Populate a stash with a collection of states.

**Parameters**

- **stash** – A stash to populate.
- **states** – A list of states with which to populate the stash.

**absorb**(*simgr*)

Collect all the states from `simgr` and put them in their corresponding stashes in this manager. This will not modify `simgr`.

**move**(*from_stash*, *to_stash*, *filter_func=None*)

Move states from one stash to another.

**Parameters**

- **from_stash** – Take matching states from this stash.
- **to_stash** – Put matching states into this stash.

- **filter_func** – Stash states that match this filter. Should be a function that takes a state and returns True or False. (default: stash all states)

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**stash**(*filter_func=None*, *from_stash='active'*, *to_stash='stashed'*)

Stash some states. This is an alias for move(), with defaults for the stashes.

**Parameters**

- **filter_func** – Stash states that match this filter. Should be a function that takes a state and returns True or False. (default: stash all states)
- **from_stash** – Take matching states from this stash. (default: 'active')
- **to_stash** – Put matching states into this stash. (default: 'stashed')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**unstash**(*filter_func=None*, *to_stash='active'*, *from_stash='stashed'*)

Unstash some states. This is an alias for move(), with defaults for the stashes.

**Parameters**

- **filter_func** – Unstash states that match this filter. Should be a function that takes a state and returns True or False. (default: unstash all states)
- **from_stash** – take matching states from this stash. (default: 'stashed')
- **to_stash** – put matching states into this stash. (default: 'active')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**drop**(*filter_func=None*, *stash='active'*)

Drops states from a stash. This is an alias for move(), with defaults for the stashes.

**Parameters**

- **filter_func** – Drop states that match this filter. Should be a function that takes a state and returns True or False. (default: drop all states)
- **stash** – Drop matching states from this stash. (default: 'active')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**apply**(*state_func=None*, *stash_func=None*, *stash='active'*, *to_stash=None*)

Applies a given function to a given stash.

**Parameters**

- **state_func** – A function to apply to every state. Should take a state and return a state. The returned state will take the place of the old state. If the function *doesn't* return a state, the old state will be used. If the function returns a list of states, they will replace the original states.
- **stash_func** – A function to apply to the whole stash. Should take a list of states and return a list of states. The resulting list will replace the stash. If both state_func and stash_func are provided state_func is applied first, then stash_func is applied on the results.
- **stash** – A stash to work with.
- **to_stash** – If specified, this stash will be used to store the resulting states instead.

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**split**(*stash_splitter=None*, *stash_ranker=None*, *state_ranker=None*, *limit=8*, *from_stash='active'*, *to_stash='stashed'*)

Split a stash of states into two stashes depending on the specified options.

The stash from_stash will be split into two stashes depending on the other options passed in. If to_stash is provided, the second stash will be written there.

stash_splitter overrides stash_ranker, which in turn overrides state_ranker. If no functions are provided, the states are simply split according to the limit.

The sort done with state_ranker is ascending.

**Parameters**

- **stash_splitter** – A function that should take a list of states and return a tuple of two lists (the two resulting stashes).
- **stash_ranker** – A function that should take a list of states and return a sorted list of states. This list will then be split according to "limit".
- **state_ranker** – An alternative to stash_splitter. States will be sorted with outputs of this function, which are to be used as a key. The first "limit" of them will be kept, the rest split off.
- **limit** – For use with state_ranker. The number of states to keep. Default: 8
- **from_stash** – The stash to split (default: 'active')
- **to_stash** – The stash to write to (default: 'stashed')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**merge**(*merge_func=None*, *merge_key=None*, *stash='active'*, *prune=True*)

Merge the states in a given stash.

**Parameters**

- **stash** – The stash (default: 'active')
- **merge_func** – If provided, instead of using state.merge, call this function with the states as the argument. Should return the merged state.

- **merge_key** – If provided, should be a function that takes a state and returns a key that will compare equal for all states that are allowed to be merged together, as a first approximation. By default: uses PC, callstack, and open file descriptors.
- **prune** – Whether to prune the stash prior to merging it

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**exception** angr.**SimulationManagerError**

Bases: *AngrError*

**class** angr.**StateHierarchy**

Bases: object

The state hierarchy holds weak references to SimStateHistory objects in a directed acyclic graph. It is useful for queries about a state's ancestry, notably "what is the best ancestor state for a merge among these states" and "what is the most recent unsatisfiable state while using LAZY_SOLVES"

**__init__**()

**get_ref**(*obj*)

**dead_ref**(*ref*)

**defer_cleanup**()

**add_state**(*s*)

**add_history**(*h*)

**simplify**()

**full_simplify**()

**lineage**(*h*)

Returns the lineage of histories leading up to *h*.

**all_successors**(*h*)

**history_successors**(*h*)

**history_predecessors**(*h*)

**history_contains**(*h*)

**unreachable_state**(*state*)

**unreachable_history**(*h*)

**most_mergeable**(*states*)

Find the "most mergeable" set of states from those provided.

**Parameters**
**states** – a list of states

**Returns**
a tuple of: (list of states to merge, those states' common history, list of states to not merge yet)

exception angr.**TracerEnvironmentError**

Bases: *AngrError*

exception angr.**UnsupportedCCallError**

Bases: *SimCCallError*, *SimUnsupportedError*

exception angr.**UnsupportedDirtyError**

Bases: *UnsupportedIRStmtError*, *SimUnsupportedError*

exception angr.**UnsupportedIRExprError**

Bases: *SimExpressionError*, *SimUnsupportedError*

exception angr.**UnsupportedIROpError**

Bases: *SimOperationError*, *SimUnsupportedError*

exception angr.**UnsupportedIRStmtError**

Bases: *SimStatementError*, *SimUnsupportedError*

exception angr.**UnsupportedNodeTypeError**

Bases: *AngrError*, NotImplementedError

angr.**UnsupportedSyscallError**

alias of *AngrUnsupportedSyscallError*

angr.**default_cc**(*arch*, *platform='Linux'*, *language=None*, *syscall=False*, *default=None*)

Return the default calling convention for a given architecture, platform, and language combination.

**Parameters**

- **arch** (str) – The architecture name.
- **platform** (str | None) – The platform name (e.g., "Linux" or "Win32").
- **language** (str | None) – The programming language name (e.g., "go").
- **syscall** (bool) – Return syscall convention (True), or normal calling convention (False, default).
- **default** (type[*SimCC*] | None) – The default calling convention to return if nothing fits.

**Return type**

type[*SimCC*] | None

**Returns**

A default calling convention class if we can find one for the architecture, platform, and language combination, or the default if nothing fits.

angr.**load_shellcode**(*shellcode*, *arch*, *start_offset=0*, *load_address=0*, *thumb=False*, ***kwargs*)

Load a new project based on a snippet of assembly or bytecode.

**Parameters**

- **shellcode** (bytes | str) – The data to load, as either a bytestring of instructions or a string of assembly text
- **arch** – The name of the arch to use, or an archinfo class
- **start_offset** – The offset into the data to start analysis (default 0)
- **load_address** – The address to place the data in memory (default 0)
- **thumb** – Whether this is ARM Thumb shellcode

angr.**register_analysis**(*cls*, *name*)

### Submodules

| | |
|---|---|
| *ail_callable* | |
| *ailment* | |
| *analyses* | |
| *angrdb* | |
| *annocfg* | |
| *blade* | |
| *block* | |
| *callable* | |
| *calling_conventions* | |
| *code_location* | |
| *codenode* | |
| *concretization_strategies* | |
| *distributed* | angr.distributed provides a simple implementation for conducting long-running symbolic-execution-based tasks. |
| *emulator* | |
| *engines* | |
| *errors* | |
| *exploration_techniques* | |
| *factory* | |
| *flirt* | |
| *keyed_region* | |
| *knowledge_base* | Representing the artifacts of a project. |
| *knowledge_plugins* | |
| *llm_client* | |
| *llm_models* | |
| *mcp* | angr MCP Server - Model Context Protocol interface for angr binary analysis. |
| *misc* | |
| *procedures* | |
| *project* | |
| *protos* | |
| *rust* | |
| *rustylib* | |
| *serializable* | |
| *sim_manager* | |
| *sim_options* | |
| *sim_procedure* | |
| *sim_state* | |
| *sim_state_options* | |
| *sim_type* | |
| *sim_variable* | |
| *simos* | Manage OS-level configuration. |
| *slicer* | |
| *state_hierarchy* | |
| *state_plugins* | |
| *storage* | |
| *tablespecs* | |
| *utils* | |
| *vaults* | |

## 10.1 angr.ail_callable

**class** angr.ail_callable.**AILCallable**

Bases: *Callable*

A Callable that uses AILCallState for its states.

**perform_call**(**args*, *prototype=None*)

## 10.2 angr.ailment

**class** angr.ailment.**AILBlockRewriter**

Bases: *AILBlockWalker*[*Expression*, *Statement*, Block]

Walks all statements and expressions of an AIL node, and rebuilds expressions, statements, or blocks if needed.

If you need a pure walker without rebuilding, use AILBlockViewer instead.

**Variables**

- **update_block** – True if the block should be updated in place, False if a new block should be created and returned as the result of walk().
- **replace_phi_stmt** – True if you want _handle_Phi be called and vvars potentially replaced; False otherwise. Default to False because in the most majority cases you do not want vvars in a Phi variable be replaced.

**__init__**(*stmt_handlers=None*, *expr_handlers=None*, *update_block=True*, *replace_phi_stmt=False*)

**Parameters**

- **update_block** (*bool*)
- **replace_phi_stmt** (*bool*)

**class** angr.ailment.**AILBlockViewer**

Bases: *AILBlockWalker*[None, None, None]

Walks all statements and expressions of an AIL node and do nothing.

**class** angr.ailment.**AILBlockWalker**

Bases: Generic

Walks all statements and expressions of an AIL node and construct arbitrary values based on them.

Note that we lazily initialize self._stmt_handlers and self._expr_handlers when they are accessed. This is to support the existing pattern of updating stmt/expr handlers in-place after creating a block walker, and is slightly slower. Overridding handler methods in a new class is the fastest approach.

**__init__**(*stmt_handlers=None*, *expr_handlers=None*)

**classmethod rebuild_default_handler_funcs**()

**Return type**
None

**property stmt_handlers:** dict[*type*, *Callable*[[int, Any, *Block* | None], StmtType]]

**property expr_handlers:** dict[*type*, *Callable*[[int, Any, int, *Statement* | None, *Block* | None], ExprType]]

**reset**()

Reset per-walk state variables so that this walker can be reused for another walk. Subclasses that updates state across a walk must override this to clear that state.

**Return type**
None

**walk**(*block*)

**Return type**
TypeVar(BlockType)

**Parameters**
**block** (Block)

**walk_statement**(*stmt*, *block=None*, *stmt_idx=0*)

**Return type**
TypeVar(StmtType)

**Parameters**

- **stmt** (Statement)
- **block** (Block | *None*)
- **stmt_idx** (*int*)

**walk_expression**(*expr*, *stmt_idx=None*, *stmt=None*, *block=None*)

**Return type**
TypeVar(ExprType)

**Parameters**

- **expr** (Expression)
- **stmt_idx** (*int* | *None*)
- **stmt** (Statement | *None*)
- **block** (Block | *None*)

**class** angr.ailment.**Assignment**

Bases: object

Marker for Statement instances whose variant is Assignment.

**static __new__**(*cls*, *idx*, *dst*, *src*, ***tags*)

**Return type**
Statement

**class** angr.ailment.**BinaryOp**

Bases: object

Marker for Expression instances whose variant is BinaryOp.

**COMPARISON_NEGATION = {'CmpEQ': 'CmpNE', 'CmpGE': 'CmpLT', 'CmpGT': 'CmpLE', 'CmpLE': 'CmpGT', 'CmpLT': 'CmpGE', 'CmpNE': 'CmpEQ'}**

static __new__(*cls*, *idx*, *op*, *operands*, *signed=False*, ** (Keyword-only parameters separator (PEP 3102))*, *bits=None*, *floating_point=False*, *rounding_mode=None*, *vector_count=None*, *vector_size=None*, ***tags*)

**Return type**
Expression

class angr.ailment.**Block**

Bases: object

classmethod __new__(**args*, ***kwargs*)

addr

clear_hash()

copy(*statements=None*)

dbg_repr(*indent=0*)

deep_copy(*manager*)

classmethod from_bytes(*data*)

Inverse of to_bytes.

idx

likes(*other*)

original_size

pp()

sort_key

statements

to_bytes()

Postcard-encode the full Block state – (addr, original_size, idx, statements), with statements embedded as their [*AilStatement*] payloads. cached_hash is transient and is recomputed after from_bytes. Every statement must be an AIL Statement; anything else raises TypeError.

class angr.ailment.**Const**

Bases: object

Marker for Expression instances whose variant is Const.

Const(idx, value, bits, **tags) returns an Expression – the marker class is not in the instance's MRO.

static __new__(*cls*, *idx*, *value*, *bits*, ***tags*)

**Return type**
Expression

class angr.ailment.**Expression**

Bases: object

Marker for isinstance(x, Expression) checks.

Real AIL expression instances are angr.rustylib.ailment.Expression (a single fat-enum pyclass). The per-variant markers (Const, BinaryOp, ...) dispatch on the variant tag via metaclass __instancecheck__.

__init__(*idx=None*, **_extra*, ***tags*)

static from_bytes(*data*)

Deserialize an Expression from bytes.

**Parameters**
**data** (bytes)

class angr.ailment.**ExpressionKind**

Bases: object

Integer tag for Expression variants. Exposed as the value of the kind getter on the Expression pyclass; replaces the prior &'static str representation so dispatch lookups become integer hash/compare instead of string interning/hash.

Note: __hash__ returns self as u64 so the enum hashes identically to its integer value – a dict keyed by ExpressionKind accepts lookups by raw int and vice versa.

**Array = <ExpressionKind.Array: 21>**

**BasePointerOffset = <ExpressionKind.BasePointerOffset: 25>**

**BinaryOp = <ExpressionKind.BinaryOp: 7>**

**Call = <ExpressionKind.Call: 14>**

**ComboRegister = <ExpressionKind.ComboRegister: 3>**

**Const = <ExpressionKind.Const: 0>**

**Convert = <ExpressionKind.Convert: 8>**

**DirtyExpression = <ExpressionKind.DirtyExpression: 15>**

**Extract = <ExpressionKind.Extract: 12>**

**FunctionLikeMacro = <ExpressionKind.FunctionLikeMacro: 24>**

**ITE = <ExpressionKind.ITE: 11>**

**Insert = <ExpressionKind.Insert: 13>**

**Let = <ExpressionKind.Let: 22>**

**Load = <ExpressionKind.Load: 10>**

**Macro = <ExpressionKind.Macro: 23>**

**MultiStatementExpression = <ExpressionKind.MultiStatementExpression: 17>**

**Phi = <ExpressionKind.Phi: 5>**

**Register = <ExpressionKind.Register: 2>**

**Reinterpret = <ExpressionKind.Reinterpret: 9>**

**RustEnum = <ExpressionKind.RustEnum: 20>**

**StackBaseOffset = <ExpressionKind.StackBaseOffset: 26>**

**StringLiteral = <ExpressionKind.StringLiteral: 18>**

**Struct = <ExpressionKind.Struct: 19>**

**Tmp = <ExpressionKind.Tmp: 1>**

**UnaryOp = <ExpressionKind.UnaryOp: 6>**

**VEXCCallExpression = <ExpressionKind.VEXCCallExpression: 16>**

**VirtualVariable = <ExpressionKind.VirtualVariable: 4>**

**name**

**value**

**class** angr.ailment.**IRSBConverter**

Bases: *Converter*

**static convert**(*irsb*, *manager*)

Convert the given IRSB to an AIL block

**Parameters**

- **irsb** – The IRSB to convert
- **manager** – The manager to use

**Returns**

Returns the converted block

**class** angr.ailment.**Manager**

Bases: object

**classmethod __new__**(**args*, ***kwargs*)

**atom_ctr**

Next atom index to hand out (the original used *itertools.count()*).

**block_addr**

**ins_addr**

**next_atom**()

**reset**()

**tyenv**

**variable_map**

Attached by Clinic so that optimization passes, peephole optimizations, and region simplifiers can use VariableMap.

**vex_stmt_idx**

**class** angr.ailment.**NoOp**

Bases: object

Marker for Statement instances whose variant is NoOp.

Placeholder for a removed statement; defines and uses no atoms. Primarily used by the AIL simplifier's dead-assignment removal so the indices of surrounding statements stay stable until the block is compacted.

**static __new__**(*cls*, *idx*, ***tags*)

**Return type**
Statement

**class** angr.ailment.**PCodeIRSBConverter**

Bases: *Converter*

Converts a p-code IRSB to an AIL block

**static convert**(*irsb*, *manager*)

Convert the given IRSB to an AIL block

**Parameters**

- **irsb** (*IRSB*) – IRSB to convert
- **manager** (Manager) – Manager to use

**Returns**
Converted block

**__init__**(*irsb*, *manager*)

**Parameters**

- **irsb** (IRSB)
- **manager** (*Manager*)

**class** angr.ailment.**Register**

Bases: object

Marker for Expression instances whose variant is Register.

**static __new__**(*cls*, *idx*, *reg_offset*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.**Statement**

Bases: object

Marker for isinstance(x, Statement) checks.

**__init__**(*idx=None*, **_extra*, ***tags*)

**static from_bytes**(*data*)

Deserialize a Statement from bytes.

**Parameters**
**data** (bytes)

**class** angr.ailment.**Tmp**

Bases: object

Marker for Expression instances whose variant is Tmp.

**static __new__**(*cls*, *idx*, *tmp_idx*, *bits*, ***tags*)

**Return type**
Expression

class angr.ailment.**UnaryOp**

Bases: object

Marker for Expression instances whose variant is UnaryOp.

**static __new__**(*cls*, *idx*, *op*, *operand*, *bits=None*, ***tags*)

**Return type**
Expression

class angr.ailment.**VEXIRSBConverter**

Bases: object

**static convert**(*irsb*, *manager*)

Fallback: convert a cached pyvex Python *IRSB* object.

**static convert_from_lift**(*arch*, *addr*, *data*, *manager*, ***, *opt_level=1*, *traceflags=0*, *strict_block_end=False*, *collect_data_refs=False*, *load_from_ro_regions=False*, *const_prop=False*, *cross_insn_opt=True*, *max_inst=99*, *max_bytes=None*, *bytes_offset=0*)

Default fast path: lift *data* at *addr* directly into libVEX and convert the resulting C IRSB to an AIL block without materializing a pyvex Python IRSB.

### Submodules

| | |
|---|---|
| *block* | Shim for from angr.ailment.block import Block. |
| *block_walker* | |
| *constant* | |
| *converter_common* | |
| *converter_pcode* | |
| *converter_vex* | |
| *expression* | AIL Expression classes. |
| *manager* | |
| *statement* | AIL Statement classes. |
| *tagged_object* | Python-side marker for the legacy isinstance(x, TaggedObject) checks. |
| *utils* | |

## 10.2.1 angr.ailment.block

Shim for from angr.ailment.block import Block.

class angr.ailment.block.**Block**

Bases: object

**classmethod __new__**(**args*, ***kwargs*)

**addr**

**clear_hash**()

**copy**(*statements=None*)

**dbg_repr**(*indent=0*)

**deep_copy**(*manager*)

**classmethod from_bytes**(*data*)

Inverse of `to_bytes`.

**idx**

**likes**(*other*)

**original_size**

**pp**()

**sort_key**

**statements**

**to_bytes**()

Postcard-encode the full Block state – `(addr, original_size, idx, statements)`, with statements embedded as their [*AilStatement*] payloads. `cached_hash` is transient and is recomputed after `from_bytes`. Every statement must be an AIL `Statement`; anything else raises `TypeError`.

### 10.2.2 angr.ailment.block_walker

**class** angr.ailment.block_walker.**AILBlockWalker**

Bases: Generic

Walks all statements and expressions of an AIL node and construct arbitrary values based on them.

Note that we lazily initialize self._stmt_handlers and self._expr_handlers when they are accessed. This is to support the existing pattern of updating stmt/expr handlers in-place after creating a block walker, and is slightly slower. Overridding handler methods in a new class is the fastest approach.

**__init__**(*stmt_handlers=None*, *expr_handlers=None*)

**classmethod rebuild_default_handler_funcs**()

**Return type**
None

**property stmt_handlers: dict[*type*, *Callable*[[int, Any, *Block* | None], StmtType]]**

**property expr_handlers: dict[*type*, *Callable*[[int, Any, int, *Statement* | None, *Block* | None], ExprType]]**

**reset**()

Reset per-walk state variables so that this walker can be reused for another walk. Subclasses that updates state across a walk must override this to clear that state.

**Return type**
None

**walk**(*block*)

**Return type**
TypeVar(BlockType)

**Parameters**
**block** (Block)

**walk_statement**(*stmt*, *block=None*, *stmt_idx=0*)

**Return type**
TypeVar(StmtType)

**Parameters**

- **stmt** (Statement)
- **block** (Block | *None*)
- **stmt_idx** (*int*)

**walk_expression**(*expr*, *stmt_idx=None*, *stmt=None*, *block=None*)

**Return type**
TypeVar(ExprType)

**Parameters**

- **expr** (Expression)
- **stmt_idx** (*int* | *None*)
- **stmt** (Statement | *None*)
- **block** (Block | *None*)

**class** angr.ailment.block_walker.**AILBlockViewer**

Bases: *AILBlockWalker*[None, None, None]

Walks all statements and expressions of an AIL node and do nothing.

**class** angr.ailment.block_walker.**AILBlockRewriter**

Bases: *AILBlockWalker*[*Expression*, *Statement*, Block]

Walks all statements and expressions of an AIL node, and rebuilds expressions, statements, or blocks if needed.

If you need a pure walker without rebuilding, use AILBlockViewer instead.

**Variables**

- **update_block** – True if the block should be updated in place, False if a new block should be created and returned as the result of walk().
- **replace_phi_stmt** – True if you want _handle_Phi be called and vvars potentially replaced; False otherwise. Default to False because in the most majority cases you do not want vvars in a Phi variable be replaced.

**__init__**(*stmt_handlers=None*, *expr_handlers=None*, *update_block=True*, *replace_phi_stmt=False*)

**Parameters**

- **update_block** (*bool*)
- **replace_phi_stmt** (*bool*)

### 10.2.3 angr.ailment.constant

### 10.2.4 angr.ailment.converter_common

**exception** angr.ailment.converter_common.**SkipConversionNotice**

Bases: Exception

**class** angr.ailment.converter_common.**Converter**

Bases: object

**static convert**(*irsb*, *manager*)

### 10.2.5 angr.ailment.converter_pcode

**class** angr.ailment.converter_pcode.**PCodeIRSBConverter**

Bases: *Converter*

Converts a p-code IRSB to an AIL block

**static convert**(*irsb*, *manager*)

Convert the given IRSB to an AIL block

**Parameters**

- **irsb** (*IRSB*) – IRSB to convert
- **manager** (Manager) – Manager to use

**Returns**
Converted block

**__init__**(*irsb*, *manager*)

**Parameters**

- **irsb** (IRSB)
- **manager** (*Manager*)

### 10.2.6 angr.ailment.converter_vex

**class** angr.ailment.converter_vex.**VEXIRSBConverter**

Bases: object

**static convert**(*irsb*, *manager*)

Fallback: convert a cached pyvex Python *IRSB* object.

**static convert_from_lift**(*arch*, *addr*, *data*, *manager*, ***, *opt_level=1*, *traceflags=0*, *strict_block_end=False*, *collect_data_refs=False*, *load_from_ro_regions=False*, *const_prop=False*, *cross_insn_opt=True*, *max_inst=99*, *max_bytes=None*, *bytes_offset=0*)

Default fast path: lift *data* at *addr* directly into libVEX and convert the resulting C IRSB to an AIL block without materializing a pyvex Python IRSB.

### 10.2.7 angr.ailment.expression

AIL Expression classes.

AIL expression instances are `angr.rustylib.ailment.Expression` – a single Rust pyclass wrapping the `AilExpression` enum. The per-variant classes in this module (`Const`, `BinaryOp`, `Load`, ...) are Python-side *marker* classes:

- Each marker's `__new__` calls one of the `Expression._new_*` factory static methods and returns the resulting `Expression` instance.
- A metaclass on each marker makes `isinstance(x, Const)` work by dispatching on the variant tag (`Expression.kind`). The `_AilMarkerMeta` checks the tag.

- For static type checkers each marker name is aliased to the `Expression` pyclass itself.
- Python-side subclassing of these markers is not supported.

**class** angr.ailment.expression.**ITE**

Bases: object

Marker for `Expression` instances whose variant is `ITE`.

**static __new__**(*cls*, *idx*, *cond*, *iftrue*, *iffalse*, ***tags*)

**Return type**
`Expression`

**class** angr.ailment.expression.**Array**

Bases: object

Marker for `Expression` instances whose variant is `Array`.

**static __new__**(*cls*, *idx*, *elements*, *bits*, ***tags*)

**Return type**
`Expression`

**class** angr.ailment.expression.**Atom**

Bases: object

Marker class for `isinstance(x, Atom)` checks.

**class** angr.ailment.expression.**BasePointerOffset**

Bases: object

Marker for `Expression` instances whose variant is `BasePointerOffset`.

**static __new__**(*cls*, *idx*, *bits*, *base*, *offset*, ***tags*)

**Return type**
`Expression`

**class** angr.ailment.expression.**BinaryOp**

Bases: object

Marker for `Expression` instances whose variant is `BinaryOp`.

**COMPARISON_NEGATION = {'CmpEQ': 'CmpNE', 'CmpGE': 'CmpLT', 'CmpGT': 'CmpLE', 'CmpLE': 'CmpGT', 'CmpLT': 'CmpGE', 'CmpNE': 'CmpEQ'}**

**static __new__**(*cls*, *idx*, *op*, *operands*, *signed=False*, ***, *bits=None*, *floating_point=False*, *rounding_mode=None*, *vector_count=None*, *vector_size=None*, ***tags*)

**Return type**
`Expression`

**class** angr.ailment.expression.**Call**

Bases: object

Marker for `Expression` instances whose variant is `Call`.

`isinstance` also matches `Macro` / `FunctionLikeMacro` instances to preserve the former subclass relationships.

**static __new__**(*cls*, *idx*, *target*, *args=None*, *bits=None*, *arg_vvars=None*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**ComboRegister**

Bases: object

Marker for Expression instances whose variant is ComboRegister.

**static __new__**(*cls*, *idx*, *registers*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**Const**

Bases: object

Marker for Expression instances whose variant is Const.

Const(idx, value, bits, **tags) returns an Expression – the marker class is not in the instance's MRO.

**static __new__**(*cls*, *idx*, *value*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**Convert**

Bases: object

Marker for Expression instances whose variant is Convert.

**TYPE_INT = <ConvertType.TYPE_INT: 0>**

**TYPE_FP = <ConvertType.TYPE_FP: 1>**

**static __new__**(*cls*, *idx*, *from_bits*, *to_bits*, *is_signed*, *operand*, *from_type=None*, *to_type=None*, *rounding_mode=None*, *vector_count=None*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**ConvertType**

Bases: object

Mirrors *angr.ailment.expression.ConvertType* (Enum).

**TYPE_FP = <ConvertType.TYPE_FP: 1>**

**TYPE_INT = <ConvertType.TYPE_INT: 0>**

**name**

**value**

**class** angr.ailment.expression.**DirtyExpression**

Bases: object

Marker for Expression instances whose variant is DirtyExpression.

**static __new__**(*cls*, *idx*, *callee*, *operands*, *, *guard=None*, *mfx=None*, *maddr=None*, *msize=None*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**Expression**

Bases: object

Marker for `isinstance(x, Expression)` checks.

Real AIL expression instances are `angr.rustylib.ailment.Expression` (a single fat-enum pyclass). The per-variant markers (Const, BinaryOp, …) dispatch on the variant tag via metaclass `__instancecheck__`.

**__init__**(*idx=None*, **_extra*, ***tags*)

**static from_bytes**(*data*)

Deserialize an Expression from bytes.

**Parameters**
**data** (bytes)

**class** angr.ailment.expression.**Extract**

Bases: object

Marker for `Expression` instances whose variant is `Extract`.

**static __new__**(*cls*, *idx*, *bits*, *base*, *offset*, *endness*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**FunctionLikeMacro**

Bases: object

Marker for `Expression` instances whose variant is `FunctionLikeMacro`.

**static __new__**(*cls*, *idx*, *name*, *args*, *bits=None*, *delimiter='()'*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**Insert**

Bases: object

Marker for `Expression` instances whose variant is `Insert`.

**static __new__**(*cls*, *idx*, *base*, *offset*, *value*, *endness*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**Let**

Bases: object

Marker for `Expression` instances whose variant is `Let`.

**static __new__**(*cls*, *idx*, *defs*, *src*, ***tags*)

**Return type**
Expression

class angr.ailment.expression.**Load**

Bases: object

Marker for Expression instances whose variant is Load.

static **__new__**(*cls*, *idx*, *addr*, *size*, *endness*, *, *guard=None*, *alt=None*, ***tags*)

**Return type**
Expression

class angr.ailment.expression.**Macro**

Bases: object

Marker for Expression instances whose variant is Macro (abstract).

Also matches FunctionLikeMacro instances to preserve the isinstance(x, Macro) -> True relationship of the former class hierarchy.

static **__new__**(*cls*, *idx*, *name*, *delimiter='()'*, ***tags*)

**Return type**
Expression

class angr.ailment.expression.**MultiStatementExpression**

Bases: object

Marker for Expression instances whose variant is MultiStatementExpression.

static **__new__**(*cls*, *idx*, *stmts*, *expr*, ***tags*)

**Return type**
Expression

class angr.ailment.expression.**Op**

Bases: object

Marker for isinstance(x, Op) checks.

class angr.ailment.expression.**Phi**

Bases: object

Marker for Expression instances whose variant is Phi.

static **__new__**(*cls*, *idx*, *bits*, *src_and_vvars*, ***tags*)

**Return type**
Expression

class angr.ailment.expression.**Register**

Bases: object

Marker for Expression instances whose variant is Register.

static **__new__**(*cls*, *idx*, *reg_offset*, *bits*, ***tags*)

**Return type**
Expression

class angr.ailment.expression.**Reinterpret**

Bases: object

Marker for Expression instances whose variant is Reinterpret.

**static __new__**(*cls*, *idx*, *from_bits*, *from_type*, *to_bits*, *to_type*, *operand*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**RustEnum**

Bases: object

Marker for Expression instances whose variant is RustEnum.

**static __new__**(*cls*, *idx*, *name*, *fields*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**StackBaseOffset**

Bases: object

Marker for Expression instances whose variant is StackBaseOffset.

**static __new__**(*cls*, *idx*, *bits*, *offset*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**StringLiteral**

Bases: object

Marker for Expression instances whose variant is StringLiteral.

**static __new__**(*cls*, *idx*, *data*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**Struct**

Bases: object

Marker for Expression instances whose variant is Struct.

**static __new__**(*cls*, *idx*, *name*, *fields*, *field_offsets*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**Tmp**

Bases: object

Marker for Expression instances whose variant is Tmp.

**static __new__**(*cls*, *idx*, *tmp_idx*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**UnaryOp**

Bases: object

Marker for Expression instances whose variant is UnaryOp.

**static __new__**(*cls*, *idx*, *op*, *operand*, *bits=None*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**VEXCCallExpression**

Bases: object

Marker for Expression instances whose variant is VEXCCallExpression.

**static __new__**(*cls*, *idx*, *callee*, *operands*, *bits*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**VirtualVariable**

Bases: object

Marker for Expression instances whose variant is VirtualVariable.

**static __new__**(*cls*, *idx*, *varid*, *bits*, *category*, *oident=None*, *reg_vvars=None*, ***tags*)

**Return type**
Expression

**class** angr.ailment.expression.**VirtualVariableCategory**

Bases: object

Mirrors *angr.ailment.expression.VirtualVariableCategory* (IntEnum).

**COMBO_REGISTER = <VirtualVariableCategory.COMBO_REGISTER: 5>**

**MEMORY = <VirtualVariableCategory.MEMORY: 2>**

**PARAMETER = <VirtualVariableCategory.PARAMETER: 3>**

**REGISTER = <VirtualVariableCategory.REGISTER: 0>**

**STACK = <VirtualVariableCategory.STACK: 1>**

**TMP = <VirtualVariableCategory.TMP: 4>**

**UNKNOWN = <VirtualVariableCategory.UNKNOWN: 6>**

**name**

**value**

angr.ailment.expression.**negate**(*expr*, *manager*)

Negate a comparison or boolean expression.

**Return type**
*Expression*

**Parameters**
**expr** (Expression)

### 10.2.8 angr.ailment.manager

class angr.ailment.manager.**Manager**

Bases: object

**classmethod __new__**(**args*, ***kwargs*)

**atom_ctr**

Next atom index to hand out (the original used *itertools.count()*).

**block_addr**

**ins_addr**

**next_atom**()

**reset**()

**tyenv**

**variable_map**

Attached by Clinic so that optimization passes, peephole optimizations, and region simplifiers can use VariableMap.

**vex_stmt_idx**

### 10.2.9 angr.ailment.statement

AIL Statement classes.

Mirror of angr.ailment.expression for the Statement side.

class angr.ailment.statement.**CAS**

Bases: object

Marker for Statement instances whose variant is CAS.

**static __new__**(*cls*, *idx*, *addr*, *data_lo*, *data_hi*, *expd_lo*, *expd_hi*, *old_lo*, *old_hi*, *endness*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**Assignment**

Bases: object

Marker for Statement instances whose variant is Assignment.

**static __new__**(*cls*, *idx*, *dst*, *src*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**ConditionalJump**

Bases: object

Marker for Statement instances whose variant is ConditionalJump.

**static __new__**(*cls*, *idx*, *condition*, *true_target*, *false_target*, ***, *true_target_idx=None*, *false_target_idx=None*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**DirtyStatement**

Bases: object

Marker for Statement instances whose variant is DirtyStatement.

static **__new__**(*cls*, *idx*, *dirty*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**Jump**

Bases: object

Marker for Statement instances whose variant is Jump.

static **__new__**(*cls*, *idx*, *target*, *target_idx=None*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**Label**

Bases: object

Marker for Statement instances whose variant is Label.

static **__new__**(*cls*, *idx*, *name*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**NoOp**

Bases: object

Marker for Statement instances whose variant is NoOp.

Placeholder for a removed statement; defines and uses no atoms. Primarily used by the AIL simplifier's dead-assignment removal so the indices of surrounding statements stay stable until the block is compacted.

static **__new__**(*cls*, *idx*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**Return**

Bases: object

Marker for Statement instances whose variant is Return.

static **__new__**(*cls*, *idx*, *ret_exprs*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**SideEffectStatement**

Bases: object

Marker for Statement instances whose variant is SideEffectStatement.

static **__new__**(*cls*, *idx*, *expr*, *ret_expr=None*, *fp_ret_expr=None*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**Statement**

Bases: object

Marker for `isinstance(x, Statement)` checks.

**__init__**(*idx=None*, **_extra*, ***tags*)

**static from_bytes**(*data*)

Deserialize a Statement from bytes.

**Parameters**
**data** (bytes)

class angr.ailment.statement.**Store**

Bases: object

Marker for `Statement` instances whose variant is `Store`.

**static __new__**(*cls*, *idx*, *addr*, *data*, *size*, *endness*, *guard=None*, ***tags*)

**Return type**
Statement

class angr.ailment.statement.**TaggedObject**

Bases: object

Marker class for backward-compatible `isinstance(x, TaggedObject)` checks.

The real hierarchy has no shared base anymore; this class only exists so legacy isinstance checks keep returning the same result.

Some legacy code does `__hash__ = ailment.statement.TaggedObject.__hash__` to opt in to the cached-hash machinery. We expose a pure-Python `__hash__` here so that idiom keeps working.

class angr.ailment.statement.**WeakAssignment**

Bases: object

Marker for `Statement` instances whose variant is `WeakAssignment`.

**static __new__**(*cls*, *idx*, *dst*, *src*, ***tags*)

**Return type**
Statement

### 10.2.10 angr.ailment.tagged_object

Python-side marker for the legacy `isinstance(x, TaggedObject)` checks.

We can get rid of this base class once IncompleteSwitchCaseHeadStatement is migrated to Rust.

class angr.ailment.tagged_object.**TagDict**

Bases: TypedDict

Schema for the `.tags` mapping on AIL data classes.

Only primitive-valued tags survive the Rust port. `reg_vvars` was promoted to a dedicated field on `VirtualVariable`. Variable information (`variable`, `variable_offset`, `reference_*`) now lives in a side `VariableMap` (see `angr.analyses.decompiler.variable_map`).

**always_propagate:** bool

**block_idx:** int

**deref_src_addr:** int

**extra_def:** bool

**extra_defs:** list[int]

**ins_addr:** int

**is_prototype_guessed:** bool

**keep_in_slice:** bool

**orig_ins_addr:** int

**reg_name:** str

**uninitialized:** bool

**vex_block_addr:** int

**vex_stmt_idx:** int

**write_size:** int

class angr.ailment.tagged_object.**TaggedObject**

Bases: object

Marker class for backward-compatible `isinstance(x, TaggedObject)` checks.

The real hierarchy has no shared base anymore; this class only exists so legacy isinstance checks keep returning the same result.

Some legacy code does `__hash__ = ailment.statement.TaggedObject.__hash__` to opt in to the cached-hash machinery. We expose a pure-Python `__hash__` here so that idiom keeps working.

**idx:** int

**tags:** Any

### 10.2.11 angr.ailment.utils

type angr.ailment.utils.**GetBitsTypeParams** = *ailment.expression.Expression*

angr.ailment.utils.**get_bits**(*expr*)

**Return type**
int

**Parameters**
**expr** (GetBitsTypeParams)

angr.ailment.utils.**is_none_or_likeable**(*arg1*, *arg2*, *is_list=False*)

Returns whether two things are both None or can like each other

angr.ailment.utils.**is_none_or_matchable**(*arg1*, *arg2*, *is_list=False*)

Returns whether two things are both None or can match each other

angr.ailment.utils.**is_lsb_extract**(*expr*)

Return `True` if `expr` is an `Extract` that takes the least-significant `expr.bits` bits of its base, considering endianness.

**Return type**
bool

**Parameters**
**expr** (Expression)

angr.ailment.utils.**is_lsb_overwrite**(*expr*)

Return `True` if `expr` is an `Insert` that overwrites the least-significant `expr.value.bits` bits of its base, considering endianness.

`Insert` counterpart of `is_lsb_extract`.

**Return type**
bool

**Parameters**
**expr** (Expression)

## 10.3 angr.analyses

**class** angr.analyses.**CDG**

Bases: *Analysis*

Implements a control dependence graph.

**__init__**(*cfg*, *start=None*, *no_construct=False*)

Constructor.

**Parameters**

- **cfg** – The control flow graph upon which this control dependence graph will build
- **start** – The starting point to begin constructing the control dependence graph
- **no_construct** – Skip the construction step. Only used in unit-testing.

**property graph**

**get_post_dominators**()

Return the post-dom tree

**get_dependants**(*run*)

Return a list of nodes that are control dependent on the given node in the control dependence graph

**get_guardians**(*run*)

Return a list of nodes on whom the specific node is control dependent in the control dependence graph

**class** angr.analyses.**CFG**

Bases: *CFGFast*

tl;dr: CFG is just a wrapper around CFGFast for compatibility issues. It will be fully replaced by CFGFast in future releases. Feel free to use CFG if you intend to use CFGFast. Please use CFGEmulated if you *have to* use the old, slow, dynamically-generated version of CFG.

For multiple historical reasons, angr's CFG is accurate but slow, which does not meet what most people expect. We developed CFGFast for light-speed CFG recovery, and renamed the old CFG class to CFGEmulated. For compatibility concerns, CFG was kept as an alias to CFGEmulated.

However, so many new users of angr would load up a binary and generate a CFG immediately after running "pip install angr", and draw the conclusion that "angr's CFG is so slow - angr must be unusable!" Therefore, we made the hard decision: CFG will be an alias to CFGFast, instead of CFGEmulated.

To ease the transition of your existing code and script, the following changes are made:

- A CFG class, which is a sub class of CFGFast, is created.
- You will see both a warning message printed out to stderr and an exception raised by angr if you are passing CFG any parameter that only CFGEmulated supports. This exception is not a sub class of AngrError, so you wouldn't capture it with your old code by mistake.
- In the near future, this wrapper class will be removed completely, and CFG will be a simple alias to CFGFast.

We expect most interfaces are the same between CFGFast and CFGEmulated. Apparently some functionalities (like context-sensitivity, and state keeping) only exist in CFGEmulated, which is when you want to use CFGEmulated instead.

**class** angr.analyses.**DDG**

Bases: *Analysis*

This is a fast data dependence graph directly generated from our CFG analysis result. The only reason for its existence is the speed. There is zero guarantee for being sound or accurate. You are supposed to use it only when you want to track the simplest data dependence, and you do not care about soundness or accuracy.

For a better data dependence graph, please consider performing a better static analysis first (like Value-set Analysis), and then construct a dependence graph on top of the analysis result (for example, the VFG in angr).

The DDG is based on a CFG, which should ideally be a CFGEmulated generated with the following options:

- keep_state=True to keep all input states
- state_add_options=angr.options.refs to store memory, register, and temporary value accesses

You may want to consider a high value for context_sensitivity_level as well when generating the CFG.

Also note that since we are using states from CFG, any improvement in analysis performed on CFG (like a points-to analysis) will directly benefit the DDG.

**__init__**(*cfg*, *start=None*, *call_depth=None*, *block_addrs=None*)

**Parameters**

- **cfg** – Control flow graph. Please make sure each node has an associated *state* with it, e.g. by passing the keep_state=True and state_add_options=angr.options.refs arguments to CFGEmulated.
- **start** – An address, Specifies where we start the generation of this data dependence graph.
- **call_depth** – None or integers. A non-negative integer specifies how deep we would like to track in the call tree. None disables call_depth limit.
- **block_addrs** (*iterable or None*) – A collection of block addresses that the DDG analysis should be performed on.

**property graph**

A networkx DiGraph instance representing the dependence relations between statements. :rtype: networkx.DiGraph

**Type**
returns

**property data_graph**

Get the data dependence graph.

**Returns**
A networkx DiGraph instance representing data dependence.

**Return type**
networkx.DiGraph

**property simplified_data_graph**

return:

**property ast_graph**

**pp()**

Pretty printing.

**dbg_repr()**

Representation for debugging.

**get_predecessors**(*code_location*)

Returns all predecessors of the code location.

**Parameters**
**code_location** – A CodeLocation instance.

**Returns**
A list of all predecessors.

**function_dependency_graph**(*func*)

Get a dependency graph for the function *func*.

**Parameters**
**func** – The Function object in CFG.function_manager.

**Returns**
A networkx.DiGraph instance.

**data_sub_graph**(*pv*, *simplified=True*, *killing_edges=False*, *excluding_types=None*)

Get a subgraph from the data graph or the simplified data graph that starts from node pv.

**Parameters**

- **pv** (*ProgramVariable*) – The starting point of the subgraph.
- **simplified** (*bool*) – When True, the simplified data graph is used, otherwise the data graph is used.
- **killing_edges** (*bool*) – Are killing edges included or not.
- **excluding_types** (*iterable*) – Excluding edges whose types are among those excluded types.

**Returns**
A subgraph.

**Return type**
networkx.MultiDiGraph

**find_definitions**(*variable*, *location=None*, *simplified_graph=True*)

Find all definitions of the given variable.

**Parameters**

- **variable** (SimVariable)
- **simplified_graph** (*bool*) – True if you just want to search in the simplified graph instead of the normal graph. Usually the simplified graph suffices for finding definitions of register or memory variables.

**Returns**
A collection of all variable definitions to the specific variable.

**Return type**
list

**find_consumers**(*var_def*, *simplified_graph=True*)

Find all consumers to the specified variable definition.

**Parameters**

- **var_def** (ProgramVariable) – The variable definition.
- **simplified_graph** (*bool*) – True if we want to search in the simplified graph, False otherwise.

**Returns**
A collection of all consumers to the specified variable definition.

**Return type**
list

**find_killers**(*var_def*, *simplified_graph=True*)

Find all killers to the specified variable definition.

**Parameters**

- **var_def** (ProgramVariable) – The variable definition.
- **simplified_graph** (*bool*) – True if we want to search in the simplified graph, False otherwise.

**Returns**
A collection of all killers to the specified variable definition.

**Return type**
list

**find_sources**(*var_def*, *simplified_graph=True*)

Find all sources to the specified variable definition.

**Parameters**

- **var_def** (ProgramVariable) – The variable definition.
- **simplified_graph** (*bool*) – True if we want to search in the simplified graph, False otherwise.

**Returns**
A collection of all sources to the specified variable definition.

**Return type**
list

**class** angr.analyses.**VFG**

Bases: *ForwardAnalysis*[*SimState*, *VFGNode*, *VFGJob*, *BlockID*, *SimState*], *Analysis*

This class represents a control-flow graph with static analysis result.

Perform abstract interpretation analysis starting from the given function address. The output is an invariant at the beginning (or the end) of each basic block.

Steps:

- Generate a CFG first if CFG is not provided.
- Identify all merge points (denote the set of merge points as Pw) in the CFG.
- Cut those loop back edges (can be derived from Pw) so that we gain an acyclic CFG.
- **Identify all variables that are 1) from memory loading 2) from initial values, or 3) phi functions. Denote**
  the set of those variables as S_{var}.
- **Start real AI analysis and try to compute a fix point of each merge point. Perform widening/narrowing only on**
  variables in S_{var}.

**__init__**(*cfg=None*, *context_sensitivity_level=2*, *start=None*, *function_start=None*, *interfunction_level=0*, *initial_state=None*, *avoid_runs=None*, *remove_options=None*, *timeout=None*, *max_iterations_before_widening=8*, *max_iterations=40*, *widening_interval=3*, *final_state_callback=None*, *status_callback=None*, *record_function_final_states=False*)

**Parameters**

- **cfg** (*CFGEmulated* | None) – The control-flow graph to base this analysis on. If none is provided, we will construct a CFGEmulated.
- **context_sensitivity_level** (int) – The level of context-sensitivity of this VFG. It ranges from 0 to infinity. Default 2.
- **function_start** (int | None) – The address of the function to analyze.
- **interfunction_level** (int) – The level of interfunction-ness to be
- **initial_state** (*SimState* | None) – A state to use as the initial one
- **avoid_runs** (list[int] | None) – A list of runs to avoid
- **remove_options** (set[str] | None) – State options to remove from the initial state. It only works when *initial_state* is None
- **timeout** (int | None)
- **final_state_callback** (Callable[[*SimState*, *CallStack*], Any] | None) – callback function when countering final state
- **status_callback** (Callable[[*VFG*], Any] | None) – callback function used in _analysis_core_baremetal
- **start** (*int | None*)
- **max_iterations_before_widening** (*int*)
- **max_iterations** (*int*)
- **widening_interval** (*int*)
- **record_function_final_states** (*bool*)

**Return type**
None

**property function_initial_states**

property function_final_states

get_any_node(*addr*)

Get any VFG node corresponding to the basic block at @addr. Note that depending on the context sensitivity level, there might be multiple nodes corresponding to different contexts. This function will return the first one it encounters, which might not be what you want.

**Return type**
*VFGNode* | None

**Parameters**
**addr** (*int*)

get_all_nodes(*addr*)

**Return type**
Generator[*VFGNode*]

irsb_from_node(*node*)

copy()

class angr.analyses.**VSA_DDG**

Bases: *Analysis*

A Data dependency graph based on VSA states. That means we don't (and shouldn't) expect any symbolic expressions.

__init__(*vfg=None*, *start_addr=None*, *interfunction_level=0*, *context_sensitivity_level=2*, *keep_data=False*)

Constructor.

**Parameters**

- **vfg** – An already constructed VFG. If not specified, a new VFG will be created with other specified parameters. *vfg* and *start_addr* cannot both be unspecified.
- **start_addr** – The address where to start the analysis (typically, a function's entry point).
- **interfunction_level** – See VFG analysis.
- **context_sensitivity_level** – See VFG analysis.
- **keep_data** – Whether we keep set of addresses as edges in the graph, or just the cardinality of the sets, which can be used as a "weight".

get_predecessors(*code_location*)

Returns all predecessors of *code_location*.

**Parameters**
**code_location** – A CodeLocation instance.

**Returns**
A list of all predecessors.

get_all_nodes(*simrun_addr*, *stmt_idx*)

Get all DDG nodes matching the given basic block address and statement index.

class angr.analyses.**AnalysesHub**

Bases: *PluginVendor*[Any]

This class contains functions for all the registered and runnable analyses,

__init__(*project*)

class angr.analyses.**Analysis**

Bases: object

This class represents an analysis on the program.

**Variables**

- **project** – The project for this analysis.
- **kb** (KnowledgeBase) – The knowledgebase object.
- **_progress_callback** – A callback function for receiving the progress of this analysis. It only takes one argument, which is a float number from 0.0 to 100.0 indicating the current progress.
- **_show_progressbar** (*bool*) – If a progressbar should be shown during the analysis. It's independent from _progress_callback.
- **_progressbar** (*progress.Progress*) – The progress bar object.

**project:** *Project*

**kb:** *KnowledgeBase*

**errors:** list[*AnalysisLogEntry*] = []

**named_errors:** defaultdict[str, list[*AnalysisLogEntry*]] = {}

**log:** list

**property ram_usage:** float

Return the current RAM usage of the Python process, in bytes. The value is updated at most once per second.

class angr.analyses.**BackwardSlice**

Bases: *Analysis*

Represents a backward slice of the program.

__init__(*cfg*, *cdg*, *ddg*, *targets=None*, *cfg_node=None*, *stmt_id=None*, *control_flow_slice=False*, *same_function=False*, *no_construct=False*)

Create a backward slice from a specific statement based on provided control flow graph (CFG), control dependence graph (CDG), and data dependence graph (DDG).

The data dependence graph can be either CFG-based, or Value-set analysis based. A CFG-based DDG is much faster to generate, but it only reflects those states while generating the CFG, and it is neither sound nor accurate. The VSA based DDG (called VSA_DDG) is based on static analysis, which gives you a much better result.

**Parameters**

- **cfg** – The control flow graph.
- **cdg** – The control dependence graph.
- **ddg** – The data dependence graph.
- **targets** – A list of "target" that specify targets of the backward slices. Each target can be a tuple in form of (cfg_node, stmt_idx), or a CodeLocation instance.
- **cfg_node** – Deprecated. The target CFGNode to reach. It should exist in the CFG.

- **stmt_id** – Deprecated. The target statement to reach.
- **control_flow_slice** – True/False, indicates whether we should slice only based on CFG. Sometimes when acquiring DDG is difficult or impossible, you can just create a slice on your CFG. Well, if you don't even have a CFG, then...
- **no_construct** – Only used for testing and debugging to easily create a BackwardSlice object.

**dbg_repr**(*max_display=10*)

Debugging output of this slice.

**Parameters**
**max_display** – The maximum number of SimRun slices to show.

**Returns**
A string representation.

**dbg_repr_run**(*run_addr*)

Debugging output of a single SimRun slice.

**Parameters**
**run_addr** – Address of the SimRun.

**Returns**
A string representation.

**annotated_cfg**(*start_point=None*)

Returns an AnnotatedCFG based on slicing result.

**is_taint_related_to_ip**(*simrun_addr*, *stmt_idx*, *taint_type*, *simrun_whitelist=None*)

Query in taint graph to check if a specific taint will taint the IP in the future or not. The taint is specified with the tuple (simrun_addr, stmt_idx, taint_type).

**Parameters**

- **simrun_addr** – Address of the SimRun.
- **stmt_idx** – Statement ID.
- **taint_type** – Type of the taint, might be one of the following: 'reg', 'tmp', 'mem'.
- **simrun_whitelist** – A list of SimRun addresses that are whitelisted, i.e. the tainted exit will be ignored if it is in those SimRuns.

**Returns**
True/False

**is_taint_impacting_stack_pointers**(*simrun_addr*, *stmt_idx*, *taint_type*, *simrun_whitelist=None*)

Query in taint graph to check if a specific taint will taint the stack pointer in the future or not. The taint is specified with the tuple (simrun_addr, stmt_idx, taint_type).

**Parameters**

- **simrun_addr** – Address of the SimRun.
- **stmt_idx** – Statement ID.
- **taint_type** – Type of the taint, might be one of the following: 'reg', 'tmp', 'mem'.
- **simrun_whitelist** – A list of SimRun addresses that are whitelisted.

**Returns**
True/False.

class angr.analyses.**BinDiff**

Bases: *Analysis*

This class computes the a diff between two binaries represented by angr Projects

**__init__**(*other_project*, *cfg_a=None*, *cfg_b=None*)

**Parameters**
**other_project** – The second project to diff

**functions_probably_identical**(*func_a_addr*, *func_b_addr*, *check_consts=False*)

Compare two functions and return True if they appear identical.

**Parameters**

- **func_a_addr** – The address of the first function (in the first binary).
- **func_b_addr** – The address of the second function (in the second binary).

**Returns**
Whether or not the functions appear to be identical.

**property identical_functions**

A list of function matches that appear to be identical

**Type**
returns

**property differing_functions**

A list of function matches that appear to differ

**Type**
returns

**differing_functions_with_consts**()

**Returns**
A list of function matches that appear to differ including just by constants

**property differing_blocks**

A list of block matches that appear to differ

**Type**
returns

**property identical_blocks**

return A list of all block matches that appear to be identical

**property blocks_with_differing_constants**

A dict of block matches with differing constants to the tuple of constants

**Type**
return

**property unmatched_functions**

**get_function_diff**(*function_addr_a*, *function_addr_b*)

**Parameters**

- **function_addr_a** – The address of the first function (in the first binary)
- **function_addr_b** – The address of the second function (in the second binary)

**Returns**
the FunctionDiff of the two functions

class angr.analyses.**BinaryOptimizer**

Bases: *Analysis*

This is a collection of binary optimization techniques we used in Mechanical Phish during the finals of Cyber Grand Challenge. It focuses on dealing with some serious speed-impacting code constructs, and *sort of* worked on *some* CGC binaries compiled with O0. Use this analysis as a reference of how to use data dependency graph and such.

There is no guarantee that BinaryOptimizer will ever work on non-CGC binaries. Feel free to give us PR or MR, but please *do not* ask for support of non-CGC binaries.

**BLOCKS_THRESHOLD = 500**

**__init__**(*cfg*, *techniques*)

**Parameters**
**techniques** (*set[str]*)

**optimize**()

class angr.analyses.**BoyScout**

Bases: *Analysis*

Try to determine the architecture and endieness of a binary blob

**__init__**(*cookiesize=1*)

class angr.analyses.**CFGArchOptions**

Bases: object

Stores architecture-specific options and settings, as well as the detailed explanation of those options and settings.

Suppose *ao* is the CFGArchOptions object, and there is an option called *ret_jumpkind_heuristics*, you can access it by *ao.ret_jumpkind_heuristics* and set its value via *ao.ret_jumpkind_heuristics = True*

**Variables**

- **OPTIONS** (*dict*) – A dict of all default options for different architectures.
- **arch** (*archinfo.Arch*) – The architecture object.
- **_options** (*dict*) – Values of all CFG options that are specific to the current architecture.

**OPTIONS = {'ARMCortexM': {'has_arm_code': (<class 'bool'>, False), 'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, False)}, 'ARMEL': {'has_arm_code': (<class 'bool'>, True), 'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, True)}, 'ARMHF': {'has_arm_code': (<class 'bool'>, True), 'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, True)}}**

**__init__**(*arch*, ***options*)

Constructor.

**Parameters**

- **arch** (*archinfo.Arch*) – The architecture instance.

- **options** (*dict*) – Architecture-specific options, which will be used to initialize this object.

**arch = None**

**class** angr.analyses.**CFGEmulated**

Bases: *ForwardAnalysis*, *CFGBase*

This class represents a control-flow graph.

**tag: str = 'CFGEmulated'**

**addr_type: Literal['int', 'block_id', 'soot'] = 'block_id'**

**__init__**(*context_sensitivity_level=1*, *start=None*, *avoid_runs=None*, *enable_function_hints=False*, *call_depth=None*, *call_tracing_filter=None*, *initial_state=None*, *starts=None*, *keep_state=False*, *indirect_jump_target_limit=100000*, *resolve_indirect_jumps=True*, *enable_advanced_backward_slicing=False*, *enable_symbolic_back_traversal=False*, *indirect_jump_resolvers=None*, *additional_edges=None*, *no_construct=False*, *normalize=False*, *max_iterations=1*, *address_whitelist=None*, *base_graph=None*, *iropt_level=None*, *max_steps=None*, *state_add_options=None*, *state_remove_options=None*, *model=None*)

All parameters are optional.

**Parameters**

- **context_sensitivity_level** – The level of context-sensitivity of this CFG (see documentation for further details). It ranges from 0 to infinity. Default 1.
- **avoid_runs** – A list of runs to avoid.
- **enable_function_hints** – Whether to use function hints (constants that might be used as exit targets) or not.
- **call_depth** – How deep in the call stack to trace.
- **call_tracing_filter** – Filter to apply on a given path and jumpkind to determine if it should be skipped when call_depth is reached.
- **initial_state** – An initial state to use to begin analysis.
- **starts** (*iterable*) – A collection of starting points to begin analysis. It can contain the following three different types of entries: an address specified as an integer, a 2-tuple that includes an integer address and a jumpkind, or a SimState instance. Unsupported entries in starts will lead to an AngrCFGError being raised.
- **keep_state** – Whether to keep the SimStates for each CFGNode.
- **resolve_indirect_jumps** – Whether to enable the indirect jump resolvers for resolving indirect jumps
- **enable_advanced_backward_slicing** – Whether to enable an intensive technique for resolving indirect jumps
- **enable_symbolic_back_traversal** – Whether to enable an intensive technique for resolving indirect jumps
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **additional_edges** – A dict mapping addresses of basic blocks to addresses of successors to manually include and analyze forward from.
- **no_construct** (*bool*) – Skip the construction procedure. Only used in unit-testing.

- **normalize** (*bool*) – If the CFG as well as all Function graphs should be normalized or not.
- **max_iterations** (*int*) – The maximum number of iterations that each basic block should be "executed". 1 by default. Larger numbers of iterations are usually required for complex analyses like loop analysis.
- **address_whitelist** (*iterable*) – A list of allowed addresses. Any basic blocks outside of this collection of addresses will be ignored.
- **base_graph** (*networkx.DiGraph*) – A basic control flow graph to follow. Each node inside this graph must have the following properties: *addr* and *size*. CFG recovery will strictly follow nodes and edges shown in the graph, and discard any control flow that does not follow an existing edge in the base graph. For example, you can pass in a Function local transition graph as the base graph, and CFGEmulated will traverse nodes and edges and extract useful information.
- **iropt_level** (*int*) – The optimization level of VEX IR (0, 1, 2). The default level will be used if *iropt_level* is None.
- **max_steps** (*int*) – The maximum number of basic blocks to recover forthe longest path from each start before pausing the recovery procedure.
- **state_add_options** – State options that will be added to the initial state.
- **state_remove_options** – State options that will be removed from the initial state.

**copy**()

Make a copy of the CFG.

**Return type**
*CFGEmulated*

**Returns**
A copy of the CFG instance.

**resume**(*starts=None*, *max_steps=None*)

Resume a paused or terminated control flow graph recovery.

**Parameters**

- **starts** (*iterable*) – A collection of new starts to resume from. If *starts* is None, we will resume CFG recovery from where it was paused before.
- **max_steps** (*int*) – The maximum number of blocks on the longest path starting from each start before pausing the recovery.

**Returns**
None

**remove_cycles**()

Forces graph to become acyclic, removes all loop back edges and edges between overlapped loop headers and their successors.

**downsize**()

Remove saved states from all CFGNodes to reduce memory usage.

**Returns**
None

**unroll_loops**(*max_loop_unrolling_times*)

Unroll loops for each function. The resulting CFG may still contain loops due to recursion, function calls, etc.

**Parameters**
**max_loop_unrolling_times** (*int*) – The maximum iterations of unrolling.

**Returns**
None

**force_unroll_loops**(*max_loop_unrolling_times*)

Unroll loops globally. The resulting CFG does not contain any loop, but this method is slow on large graphs.

**Parameters**
**max_loop_unrolling_times** (*int*) – The maximum iterations of unrolling.

**Returns**
None

**immediate_dominators**(*start*, *target_graph=None*)

Get all immediate dominators of sub graph from given node upwards.

**Parameters**

- **start** (*str*) – id of the node to navigate forwards from.
- **target_graph** (*networkx.classes.digraph.DiGraph*) – graph to analyse, default is self.graph.

**Returns**
each node of graph as index values, with element as respective node's immediate dominator.

**Return type**
dict

**immediate_postdominators**(*end*, *target_graph=None*)

Get all immediate postdominators of sub graph from given node upwards.

**Parameters**

- **start** (*str*) – id of the node to navigate forwards from.
- **target_graph** (*networkx.classes.digraph.DiGraph*) – graph to analyse, default is self.graph.

**Returns**
each node of graph as index values, with element as respective node's immediate dominator.

**Return type**
dict

**remove_fakerets**()

Get rid of fake returns (i.e., Ijk_FakeRet edges) from this CFG

**Returns**
None

**get_topological_order**(*cfg_node*)

Get the topological order of a CFG Node.

**Parameters**
**cfg_node** – A CFGNode instance.

**Returns**
An integer representing its order, or None if the CFGNode does not exist in the graph.

`get_subgraph`(*starting_node*, *block_addresses*)

Get a sub-graph out of a bunch of basic block addresses.

**Parameters**

- **starting_node** (CFGNode) – The beginning of the subgraph
- **block_addresses** (*iterable*) – A collection of block addresses that should be included in the subgraph if there is a path between *starting_node* and a CFGNode with the specified address, and all nodes on the path should also be included in the subgraph.

**Returns**
A new CFG that only contain the specific subgraph.

**Return type**
*CFGEmulated*

`get_function_subgraph`(*start*, *max_call_depth=None*)

Get a sub-graph of a certain function.

**Parameters**

- **start** – The function start. Currently it should be an integer.
- **max_call_depth** – Call depth limit. None indicates no limit.

**Returns**
A CFG instance which is a sub-graph of self.graph

`property context_sensitivity_level`

`property graph: SpillingCFG`

`property unresolvables`

Get those SimRuns that have non-resolvable exits.

**Returns**
A set of SimRuns

**Return type**
set

`property deadends`

Get all CFGNodes that has an out-degree of 0

**Returns**
A list of CFGNode instances

**Return type**
list

`class angr.analyses.CFGFast`

Bases: *ForwardAnalysis*[*CFGNode*, *CFGNode*, *CFGJob*, int, object], *CFGBase*

We find functions inside the given binary, and build a control-flow graph in very fast manners: instead of simulating program executions, keeping track of states, and performing expensive data-flow analysis, CFGFast will only perform light-weight analyses combined with some heuristics, and with some strong assumptions.

In order to identify as many functions as possible, and as accurate as possible, the following operation sequence is followed:

# Active scanning

- If the binary has "function symbols" (TODO: this term is not accurate enough), they are starting points of the code scanning
- If the binary does not have any "function symbol", we will first perform a function prologue scanning on the entire binary, and start from those places that look like function beginnings
- Otherwise, the binary's entry point will be the starting point for scanning

# Passive scanning

- After all active scans are done, we will go through the whole image and scan all code pieces

Due to the nature of those techniques that are used here, a base address is often not required to use this analysis routine. However, with a correct base address, CFG recovery will almost always yield a much better result. A custom analysis, called GirlScout, is specifically made to recover the base address of a binary blob. After the base address is determined, you may want to reload the binary with the new base address by creating a new Project object, and then re-recover the CFG.

```
PRINTABLES = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\
'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r'

SPECIAL_THUNKS = {'AMD64':
{b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x89\x04$\xc3':  ('jmp', 'rax'),
b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x8dd$\x08\xc3':  ('ret',)}}

tag:  str = 'CFGFast'

addr_type:  Literal['int', 'block_id', 'soot'] = 'int'
```

__init__(*binary=None*, *objects=None*, *regions=None*, *pickle_intermediate_results=False*, *symbols=True*, *function_prologues=None*, *resolve_indirect_jumps=True*, *force_segment=False*, *force_smart_scan=None*, *force_complete_scan=False*, *indirect_jump_target_limit=100000*, *data_references=True*, *cross_references=False*, *normalize=False*, *start_at_entry=True*, *function_starts=None*, *extra_memory_regions=None*, *data_type_guessing_handlers=None*, *arch_options=None*, *indirect_jump_resolvers=None*, *base_state=None*, *exclude_sparse_regions=True*, *skip_specific_regions=True*, *heuristic_plt_resolving=None*, *detect_tail_calls=False*, *low_priority=False*, *cfb=None*, *model=None*, *resume_state=None*, *eh_frame=True*, *eh_frame_boundaries=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=2048*, *nodecode_threshold=0.6*, *nodecode_step=16483*, *repeating_byte_run_threshold=64*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=True*, *treat_functions_as_complete=True*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=None*, *elf_eh_frame=None*, ***extra_arch_options*)

**Parameters**

- **binary** – The binary to recover CFG on. By default the main binary is used.
- **objects** – A list of objects to recover the CFG on. By default it will recover the CFG of all loaded objects.
- **regions** (*iterable*) – A list of tuples in the form of (start address, end address) describing memory regions that the CFG should cover.
- **pickle_intermediate_results** (*bool*) – If we want to store the intermediate results or not.
- **symbols** (*bool*) – Get function beginnings from symbols in the binary.

- **function_prologues** (bool | None) – Scan the binary for function prologues, and use those positions as function beginnings
- **resolve_indirect_jumps** (*bool*) – Try to resolve indirect jumps. This is necessary to resolve jump targets from jump tables, etc.
- **force_segment** (*bool*) – Force CFGFast to rely on binary segments instead of sections.
- **force_complete_scan** (*bool*) – Perform a complete scan on the binary and maximize the number of identified code blocks.
- **data_references** (*bool*) – Enables the collection of references to data used by individual instructions. This does not collect 'cross-references', particularly those that involve multiple instructions. For that, see *cross_references*
- **cross_references** (*bool*) – Whether CFGFast should collect "cross-references" from the entire program or not. This will populate the knowledge base with references to and from each recognizable address constant found in the code. Note that, because this performs constant propagation on the entire program, it may be much slower and consume more memory. This option implies *data_references=True*.
- **normalize** (*bool*) – Normalize the CFG as well as all function graphs after CFG recovery.
- **start_at_entry** (*bool*) – Begin CFG recovery at the entry point of this project. Setting it to False prevents CFGFast from viewing the entry point as one of the starting points of code scanning.
- **function_starts** (*list*) – A list of extra function starting points. CFGFast will try to resume scanning from each address in the list.
- **extra_memory_regions** (*list*) – A list of 2-tuple (start-address, end-address) that shows extra memory regions. Integers falling inside will be considered as pointers.
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **base_state** – A state to use as a backer for all memory loads
- **detect_tail_calls** (*bool*) – Enable aggressive tail-call optimization detection.
- **eh_frame** (*bool*) – Retrieve function starts (and maybe sizes later) from the .eh_frame of ELF binaries or exception records of PE binaries.
- **eh_frame_boundaries** (*bool*) – Treat function starts recorded in the .eh_frame of ELF binaries as hard function boundaries while tracing, the same way symbol addresses are treated. Without it, a tail-called function that is reached through a jump before it is reached through a call is absorbed into the caller. Only used if eh_frame is enabled.
- **skip_unmapped_addrs** – Ignore all branches into unmapped regions. True by default. You may want to set it to False if you are analyzing manually patched binaries or malware samples.
- **indirect_calls_always_return** (bool | None) – Should CFG assume indirect calls must return or not. Assuming indirect calls must return will significantly reduce the number of constant propagation runs, but may reduce the overall CFG recovery precision when facing non-returning indirect calls. By default, we only assume indirect calls always return for large binaries (region > 50KB).
- **jumptable_resolver_resolves_calls** (bool | None) – Whether JumpTableResolver should resolve indirect calls or not. Most indirect calls in C++ binaries or UEFI binaries

cannot be resolved using jump table resolver and must be resolved using their specific resolvers. By default, we will only disable JumpTableResolver from resolving indirect calls for large binaries (region > 50 KB).

- **repeating_byte_run_threshold** – The minimum length of a run of one repeated byte that CFGFast refuses to decode as code. Such runs are filler, but many of them decode cleanly (e.g., 0x91 is *xchg ecx, eax* on x86), so linear disassembly would otherwise walk through them forever. Runs of the architecture's nop byte are exempt because execution really does flow through them. Set it to 0 to disable the check.
- **check_funcret_max_job** – When popping return-site jobs out of the job queue, angr will prioritize jobs for which the callee is known to return. This check may be slow when there are a large amount of jobs in different caller functions, and this situation often occurs in obfuscated binaries where many functions never return. This parameter acts as a threshold to disable this check when the number of jobs in the queue exceeds this threshold.
- **start** (*int*) – (Deprecated) The beginning address of CFG recovery.
- **end** (*int*) – (Deprecated) The end address of CFG recovery.
- **arch_options** (CFGArchOptions) – Architecture-specific options.
- **extra_arch_options** – Any key-value pair in kwargs will be seen as an arch-specific option and will be used to set the option value in self._arch_options.
- **retedges** (bool) – Whether to add return edges (from function endpoints to their return sites) in the CFG. Return edges are not added by default because they are often not useful during analysis; You can set retedges to True or call make_return_edges() after CFG recovery to create return edges. Note that this option does not impact function graphs.
- **model** – An existing CFGModel to write into. When a partial model from a previously aborted CFG recovery is passed in, recovery resumes on top of it: bytes covered by existing nodes and memory data are treated as already scanned and will not be re-lifted.
- **treat_functions_as_complete** (bool) – Whether post-analysis may treat the recovered functions as complete (drop bad functions, conclude returning=False for functions without return paths). Set to False for incremental runs that extend a partial model without completing it (see the resume recipes below).
- **resume_state** (*CFGResumeState* | None) – The resume_state captured by a previously aborted CFGFast instance. Its unprocessed jobs and unresolved indirect jumps are recreated (with their original function context) so that the recovery continues where it left off. Pass it together with model=.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress. The callback receives (percentage, text=..., cfg=...) where cfg is this CFGFast instance; calling cfg.abort() from the callback gracefully aborts the analysis: the core recovery loop stops at the next job boundary, post-analysis still runs and finalizes the partially recovered model, and cfg.should_abort remains True so callers can tell the model is partial.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.
- **force_smart_scan** (*bool | None*)
- **drop_bad_funcs** (*bool*)

Aborting and resuming CFG recovery:

After a graceful abort (see progress_callback above), self.resume_state captures the jobs that were still unprocessed (with their function context) and the unresolved indirect jumps; self.unprocessed_job_addrs

holds the corresponding addresses for informational purposes. To resume recovery, create a new CFGFast instance reusing the partial model. To resume global scanning and finish the entire binary, re-inject the captured state while keeping the seeding options (symbols, function_prologues, eh_frame, force_smart_scan) at their original values:

```
proj.analyses.CFGFast(model=partial_cfg.model, start_at_entry=False,
                      resume_state=partial_cfg.resume_state)
```

Do not pass unprocessed_job_addrs as function_starts instead: function_starts seeds are treated as function heads, which would promote unprocessed call return sites and intra-function jump targets to functions.

To only resume from specific addresses (they must not be part of the partial model already; seeding an already-scanned address is a no-op):

```
proj.analyses.CFGFast(model=partial_cfg.model, function_starts=[addr], start_at_
↪entry=False,
                      symbols=False, function_prologues=False, eh_frame=False,
                      force_smart_scan=False, force_complete_scan=False,
                      treat_functions_as_complete=False)
```

treat_functions_as_complete=False tells post-analysis that the resulting model is still partial: functions must not be dropped as bad, and returning=False must not be concluded for (possibly truncated) functions - such conclusions would poison a later full resume the same way an abort would without them being reset.

**Returns**
None

**Parameters**

- **function_prologues** (*bool | None*)
- **force_smart_scan** (*bool | None*)
- **resume_state** (CFGResumeState | *None*)
- **indirect_calls_always_return** (*bool | None*)
- **jumptable_resolver_resolves_calls** (*bool | None*)
- **retedges** (*bool*)
- **drop_bad_funcs** (*bool*)
- **treat_functions_as_complete** (*bool*)

**property graph:** *SpillingCFG*

**property memory_data**

**property jump_tables**

**property insn_addr_to_memory_data**

**do_full_xrefs**(*overlay_state=None*)

Perform xref recovery on all functions.

**Parameters**
**overlay** (SimState) – An overlay state for loading constant data.

**Returns**
None

**drop_bad_functions**()

**make_return_edges**()

For each returning function, create return edges in self.graph.

**Returns**
None

**copy**()

**output**()

class angr.analyses.**CFGFastSoot**

Bases: *CFGFast*

**addr_type**: Literal['int', 'block_id', 'soot'] = 'soot'

**drop_bad_functions**()

**make_functions**()

Revisit the entire control flow graph, create Function instances accordingly, and correctly put blocks into each function.

Although Function objects are crated during the CFG recovery, they are neither sound nor accurate. With a pre-constructed CFG, this method rebuilds all functions bearing the following rules:

- A block may only belong to one function.
- Small functions lying inside the startpoint and the endpoint of another function will be merged with the other function
- Tail call optimizations are detected.
- PLT stubs are aligned by 16.

**Returns**
None

class angr.analyses.**CalleeCleanupFinder**

Bases: *Analysis*

**__init__**(*starts=None*, *hook_all=False*)

**analyze**(*addr*)

class angr.analyses.**CallingConventionAnalysis**

Bases: *Analysis*

Analyze the calling convention of a function and guess a probable prototype.

The calling convention of a function can be inferred at both its call sites and the function itself. At call sites, we consider all register and stack variables that are not alive after the function call as parameters to this function. In the function itself, we consider all register and stack variables that are read but without initialization as parameters. Then we synthesize the information from both locations and make a reasonable inference of calling convention of this function.

**Variables**

- **_function** – The function to recover calling convention for.
- **_variable_manager** – A handy accessor to the variable manager.

- **_cfg** – A reference of the CFGModel of the current binary. It is used to discover call sites of the current function in order to perform analysis at call sites.
- **analyze_callsites** – True if we should analyze all call sites of the current function to determine the calling convention and arguments. This can be time-consuming if there are many call sites to analyze.
- **cc** (*SimCC* | None) – The recovered calling convention for the function.
- **_collect_facts** – True if we should run FunctionFactCollector to collect input arguments and return value size. False if input arguments and return value size are provided by the user.

**__init__**(*func*, *cfg=None*, *analyze_callsites=False*, *caller_func_addr=None*, *callsite_block_addr=None*, *callsite_insn_addr=None*, *func_graph=None*, *input_args=None*, *retval_size=None*, *extra_pop=None*, *collect_facts=False*, *collect_facts_arg_uses=False*, *collect_facts_arg_passthru=False*)

**Parameters**

- **func** (Function | *int* | *str* | *None*)
- **cfg** (CFGModel | *None*)
- **analyze_callsites** (*bool*)
- **caller_func_addr** (*int* | *None*)
- **callsite_block_addr** (*int* | *None*)
- **callsite_insn_addr** (*int* | *None*)
- **func_graph** (*DiGraph* | *None*)
- **input_args** (*list[SimRegArg* | *SimStackArg]* | *None*)
- **retval_size** (*int* | *None*)
- **extra_pop** (*int* | *None*)
- **collect_facts** (*bool*)
- **collect_facts_arg_uses** (*bool*)
- **collect_facts_arg_passthru** (*bool*)

**is_va_start_amd64**(*func*)

**Return type**
tuple[bool, int | None]

**Parameters**
**func** (Function)

**class** angr.analyses.**ClassIdentifier**

Bases: *Analysis*

This is a class identifier for non stripped or partially stripped binaries, it identifies classes based on the demangled function names, and also assigns functions to their respective classes based on their names. It also uses the results from the VtableFinder analysis to assign the corresponding vtable to the classes.

self.classes contains a mapping between class names and SimCppClass objects

e.g. A::tool() and A::qux() belong to the class A

**__init__**()

class angr.analyses.**CodeCaveAnalysis**

Bases: *Analysis*

Best-effort static location of potential vacant code caves for possible code injection: - Padding functions - Unreachable code

**__init__**()

**codecaves**: list[*CodeCave*]

class angr.analyses.**CodeTagging**

Bases: *Analysis*

**__init__**(*func*)

**analyze**()

**has_xor**()

Detects if there is any xor operation in the function.

**Returns**
Tags

**has_bitshifts**()

Detects if there is any bitwise operation in the function.

**Returns**
Tags.

**has_sql**()

Detects if there is any reference to strings that look like SQL queries.

class angr.analyses.**CompleteCallingConventionsAnalysis**

Bases: *Analysis*

Implements full-binary calling convention analysis. During the initial analysis of a binary, you may set *recover_variables* to True so that it will perform variable recovery on each function before performing calling convention analysis.

**__init__**(*mode=CallingConventionAnalysisMode.FASTISH*, *recover_variables=False*, *low_priority=False*, *force=False*, *cfg=None*, *analyze_callsites=False*, *skip_signature_matched_functions=False*, *max_function_blocks=None*, *max_function_size=None*, *workers=0*, *cc_callback=None*, *prioritize_func_addrs=None*, *skip_other_funcs=False*, *auto_start=True*, *func_graphs=None*, *target_functions=None*)

**Parameters**

- **recover_variables** – Recover variables on each function before performing calling convention analysis.
- **low_priority** – Run in the background - periodically release GIL.
- **force** – Perform calling convention analysis on functions even if they have calling conventions or prototypes already specified (or previously recovered).
- **cfg** (*CFGFast* | *CFGModel* | None) – The control flow graph model, which will be passed to CallingConventionAnalysis.
- **analyze_callsites** (bool) – Consider artifacts at call sites when performing calling convention analysis.

- **skip_signature_matched_functions** (bool) – Do not perform calling convention analysis on functions that match against existing FLIRT signatures.
- **max_function_blocks** (int | None) – Do not perform calling convention analysis on functions with more than the specified number of blocks. Setting it to None disables this check.
- **max_function_size** (int | None) – Do not perform calling convention analysis on functions whose sizes are more than *max_function_size*. Setting it to None disables this check.
- **workers** (int) – Number of multiprocessing workers.
- **mode** (CallingConventionAnalysisMode)
- **cc_callback** (*Callable | None*)
- **prioritize_func_addrs** (*list[int] | set[int] | None*)
- **skip_other_funcs** (*bool*)
- **auto_start** (*bool*)
- **func_graphs** (*dict[int, DiGraph] | None*)
- **target_functions** (*set[int] | None*)

**work**()

**prioritize_functions**(*func_addrs_to_prioritize*)

Prioritize the analysis of specified functions.

**Parameters**

**func_addrs_to_prioritize** (Iterable[int]) – A collection of function addresses to analyze first.

**static function_needs_variable_recovery**(*func*)

Check if running variable recovery on the function is the only way to determine the calling convention of the this function.

We do not need to run variable recovery to determine the calling convention of a function if: - The function is a SimProcedure. - The function is a PLT stub. - The function is a library function and we already know its prototype.

**Parameters**

**func** – The function object.

**Returns**

True if we must run VariableRecovery before we can determine what the calling convention of this function is. False otherwise.

**Return type**

bool

**class** angr.analyses.**CongruencyCheck**

Bases: *Analysis*

This is an analysis to ensure that angr executes things identically with different execution backends (i.e., unicorn vs vex).

**__init__**(*throw=False*)

Initializes a CongruencyCheck analysis.

**Parameters**

**throw** – whether to raise an exception if an incongruency is found.

**set_state_options**(*left_add_options=None*, *left_remove_options=None*, *right_add_options=None*, *right_remove_options=None*)

Checks that the specified state options result in the same states over the next *depth* states.

**set_states**(*left_state*, *right_state*)

Checks that the specified paths stay the same over the next *depth* states.

**set_simgr**(*simgr*)

**run**(*depth=None*)

Checks that the paths in the specified path group stay the same over the next *depth* bytes.

The path group should have a "left" and a "right" stash, each with a single path.

**compare_path_group**(*pg*)

**compare_states**(*sl*, *sr*)

Compares two states for similarity.

**compare_paths**(*pl*, *pr*)

**class** angr.analyses.**DataDependencyGraphAnalysis**

Bases: *Analysis*

This is a DYNAMIC data dependency graph that utilizes a given SimState to produce a DDG graph that is accurate to the path the program took during execution.

This analysis utilizes the SimActionData objects present in the provided SimState's action history to generate the dependency graph.

**__init__**(*end_state*, *start_from=None*, *end_at=None*, *block_addrs=None*)

**Parameters**

- **end_state** (*SimState*) – Simulation state used to extract all SimActionData
- **start_from** (int | None) – An address or None, Specifies where to start generation of DDG
- **end_at** (int | None) – An address or None, Specifies where to end generation of DDG
- **block_addrs** (list[int] | None) – List of block addresses that the DDG analysis should be run on

**property graph: DiGraph | None**

**property simplified_graph: DiGraph | None**

**property sub_graph: DiGraph | None**

**get_data_dep**(*g_node*, *include_tmp_nodes*, *backwards*)

**Return type**
DiGraph | None

**Parameters**

- **g_node** (BaseDepNode)
- **include_tmp_nodes** (*bool*)
- **backwards** (*bool*)

**class** angr.analyses.**Decompiler**

Bases: *Analysis*

The decompiler analysis.

Run this on a Function object for which a normalized CFG has been constructed. The fully processed output can be found in result.codegen.text

AIL graphs exposed on the result (both on a fresh run and on a cache hit, including caches reloaded from angrdb or the runtime-db spill):

- `ail_graph` (= `clinic.cc_graph`): the simplified graph before region identification.
- `clinic.graph`: the final graph after region identification and region simplification.
- `unoptimized_ail_graph` (= `clinic.unoptimized_graph`): a snapshot before the first structure-altering optimization pass; use it for an exact instruction-to-AIL mapping. Only built when `save_unoptimized_graph=True` is passed; otherwise this attribute is None on both fresh runs and cache hits.

Complexity guard: pathologically large functions (e.g. packer filler that linear disassembly turns into thousands of blocks) can keep the pipeline busy indefinitely, so two size limits are checked before the expensive work starts. `max_function_blocks` (default 50000, `DEFAULT_MAX_FUNCTION_BLOCKS`) bounds the number of CFG basic blocks and is checked before any AIL is built; `max_ail_statements` (default 1000000, `DEFAULT_MAX_AIL_STATEMENTS`) bounds the number of AIL statements and is checked right after the AIL graph is built. Both are ~10x the largest function in angr's test binary corpus. Either limit can be changed through the decompilation option of the same name, and setting one to 0 disables it. When a limit is exceeded, decompilation stops with an `AngrDecompilationComplexityError` naming the limit and the actual size: with `fail_fast=True` it is raised, otherwise it is recorded in `self.errors` (and in `kb.decompilations[...].errors`) and exposed as `complexity_error`, and `codegen` stays None.

The optimization passes come from `preset` (a name or a DecompilationPreset), or from an explicit `optimization_passes` list when no preset is given. `disable_opts` drops individual passes from the preset's list, which is how a caller asks for "this preset, minus that one pass" without restating it.

**__init__**(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *disable_opts=None*, *sp_tracker_track_memory=True*, *peephole_optimizations=None*, *vars_must_struct=None*, *flavor='pseudocode'*, *expr_comments=None*, *stmt_comments=None*, *ite_exprs=None*, *binop_operators=None*, *decompile=True*, *regen_clinic=False*, *inline_functions=None*, *desired_variables=None*, *update_memory_data=True*, *want_full_graph=False*, *generate_code=True*, *use_cache=True*, *update_cache=True*, *expr_collapse_depth=16*, *clinic_graph=None*, *clinic_arg_vvars=None*, *clinic_start_stage=None*, *clinic_end_stage=None*, *clinic_skip_stages=()*, *static_vvars=None*, *static_buffers=None*, *codegen_cls=<class 'angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator'>*, *save_unoptimized_graph=False*)

**Parameters**

- **func** (Function | *str* | *int*)
- **cfg** (CFGFast | CFGModel | *None*)
- **preset** (*str* | DecompilationPreset | *None*)
- **peephole_optimizations** (*Iterable*[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase]] | *None*)
- **vars_must_struct** (*set*[*str*] | *None*)
- **update_memory_data** (*bool*)
- **want_full_graph** (*bool*)

- **generate_code** (*bool*)
- **use_cache** (*bool*)
- **update_cache** (*bool*)
- **expr_collapse_depth** (*int*)
- **static_vvars** (*dict | None*)
- **static_buffers** (*dict | None*)
- **save_unoptimized_graph** (*bool*)

**property complexity_error:** *AngrDecompilationComplexityError* | None

The complexity-limit violation that aborted this decompilation, or None if no limit was exceeded. Inspect `limit_name`, `limit`, and `actual` on it to decide whether to raise the limit or skip the function.

**reflow_variable_types**(*cache*)

Re-run type inference on an existing variable recovery result, then rerun codegen to generate new results.

**Returns**

**Parameters**
**cache** (DecompilationCache)

**find_data_references_and_update_memory_data**(*seq_node*)

**Parameters**
**seq_node** (SequenceNode)

**transform_graph_from_ssa**(*ail_graph*)

Translate an SSA AIL graph out of SSA form. This is useful for producing a non-SSA AIL graph for displaying in angr management.

**Parameters**
**ail_graph** (DiGraph) – The AIL graph to transform out of SSA form.

**Return type**
DiGraph

**Returns**
The translated AIL graph.

**transform_seqnode_from_ssa**(*seq_node*)

**Return type**
*SequenceNode*

**Parameters**
**seq_node** (SequenceNode)

**llm_refine**()

Use the configured LLM to suggest improved variable names, function names, and variable types. Returns True if any changes were made.

**Return type**
bool

**llm_suggest_variable_names**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest better variable names for the decompiled code. Returns True if any variables were renamed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller. If False (default), exceptions are caught and the method returns False.
- **code_text** (*str | None*)

**Return type**
bool

**llm_suggest_function_name**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest a better function name. Only suggests rename for auto-generated names (starting with `sub_` or `fcn.`). Returns True if the function was renamed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
bool

**llm_suggest_variable_types**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest better C types for variables. Returns True if any variable types were changed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
bool

**llm_summarize_function**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to produce a natural-language summary of what the decompiled function does. The summary is stored in the DecompilationCache and returned.

Returns the summary string, or None if summarization failed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
str | None

**static options_to_params**(*options*)

Convert decompilation options to a dict of params.

**Parameters**
**options** (list[tuple[*DecompilationOption*, Any]]) – The decompilation options.

**Return type**
dict[str, Any]

**Returns**
A dict of keyword arguments.

**class** angr.analyses.**Disassembly**

Bases: *Analysis*

Produce formatted machine code disassembly.

__init__(*function=None*, *ranges=None*, *thumb=False*, *include_ir=False*, *block_bytes=None*)

**Parameters**

- **function** (Function | *None*)
- **ranges** (*Sequence[tuple[int, int]] | None*)
- **thumb** (*bool*)
- **include_ir** (*bool*)
- **block_bytes** (bytes | *None*)

func_lookup(*block*)

parse_block(*block*)

Parse instructions for a given block node

**Return type**
None

**Parameters**
**block** (BlockNode)

render(*formatting=None*, *show_edges=True*, *show_addresses=True*, *show_bytes=False*, *ascii_only=None*, *color=True*, *min_edge_depth=0*)

Render the disassembly to a string, with optional edges and addresses.

Color will be added by default, if enabled. To disable color pass an empty formatting dict.

**Return type**
str

**Parameters**

- **show_edges** (*bool*)
- **show_addresses** (*bool*)
- **show_bytes** (*bool*)
- **ascii_only** (*bool | None*)
- **color** (*bool*)
- **min_edge_depth** (*int*)

class angr.analyses.**DominanceFrontier**

Bases: Generic

Computes the dominance frontier of all nodes in a function graph, and provides an easy-to-use interface for querying the frontier information.

__init__(*func*, *func_graph=None*, *entry=None*, *exception_edges=False*)

**Overloads**

- **self**, **func** (Function), **func_graph** (networkx.DiGraph[T_co]), **entry** (T_co), **exception_edges** (bool)
- **self** (DominanceFrontier[CodeNode]), **func** (Function), **func_graph** (networkx.DiGraph[CodeNode] | None), **entry** (CodeNode | None), **exception_edges** (bool)

class angr.analyses.**FactCollector**

Bases: *Analysis*

An extremely fast analysis that extracts necessary facts of a function for CallingConventionAnalysis to make decision on the calling convention and prototype of a function.

**__init__**(*func*, *max_depth=100*, *track_arg_uses=False*, *track_arg_passthru=False*)

**Parameters**

- **func** (Function)
- **max_depth** (*int*)
- **track_arg_uses** (*bool*)
- **track_arg_passthru** (*bool*)

**retval_incidental: bool**

True when every endpoint's write to the return register looks like a leftover rather than a return value: the value written is consumed again in the same block (stored, tested, passed on) after the write. A void function whose last instruction happens to load into rax looks exactly like a getter from the outside; this is the one local hint that it is not, and CallingConventionAnalysis only lets call-site evidence demote a prototype to void when it is set.

class angr.analyses.**FastConstantPropagation**

Bases: *Analysis*

An extremely fast constant propagation analysis that finds function-wide constant values with potentially high false negative rates.

**__init__**(*func*, *blocks=None*, *vex_cross_insn_opt=False*, *load_callback=None*)

**Parameters**

- **func** (Function)
- **blocks** (*set[Block] | None*)
- **vex_cross_insn_opt** (*bool*)
- **load_callback** (*Callable | None*)

class angr.analyses.**FlirtAnalysis**

Bases: *Analysis*

FlirtAnalysis accomplishes two purposes:

- If a FLIRT signature file is specified, it will match the given signature file against the current binary and rename recognized functions accordingly.
- If no FLIRT signature file is specified, it will use strings to determine possible libraries embedded in the current binary, and then match all possible signatures for the architecture.

**__init__**(*sig=None*, *max_mismatched_bytes=0*, *dry_run=False*, *match_named_functions=False*)

**Parameters**

- **sig** (FlirtSignature | *str* | *None*)
- **max_mismatched_bytes** (*int*)
- **dry_run** (*bool*)
- **match_named_functions** (*bool*)

class angr.analyses.**ForwardAnalysis**

Bases: Generic

This is my very first attempt to build a static forward analysis framework that can serve as the base of multiple static analyses in angr, including CFG analysis, VFG analysis, DDG, etc.

In short, ForwardAnalysis performs a forward data-flow analysis by traversing a graph, compute on abstract values, and store results in abstract states. The user can specify what graph to traverse, how a graph should be traversed, how abstract values and abstract states are defined, etc.

ForwardAnalysis has a few options to toggle, making it suitable to be the base class of several different styles of forward data-flow analysis implementations.

ForwardAnalysis supports a special mode when no graph is available for traversal (for example, when a CFG is being initialized and constructed, no other graph can be used). In that case, the graph traversal functionality is disabled, and the optimal graph traversal order is not guaranteed. The user can provide a job sorting method to sort the jobs in queue and optimize traversal order.

Feel free to discuss with me (Fish) if you have any suggestions or complaints.

**__init__**(*order_jobs=False*, *allow_merging=False*, *allow_widening=False*, *status_callback=None*, *graph_visitor=None*)

Constructor

**Parameters**

- **order_jobs** (*bool*) – If all jobs should be ordered or not.
- **allow_merging** (*bool*) – If job merging is allowed.
- **allow_widening** (*bool*) – If job widening is allowed.
- **graph_visitor** (Optional[*GraphVisitor*[TypeVar(NodeType)]]) – A graph visitor to provide successors.
- **status_callback** (Callable[[ForwardAnalysis], *Any*] | *None*)

**Returns**
None

**property should_abort**

Should the analysis be terminated. :return: True/False

**property graph: DiGraph**

**property jobs**

**abort**()

Abort the analysis :return: None

**has_job**(*job*)

Checks whether there exists another job which has the same job key. :type job: TypeVar(JobType) :param job: The job to check.

**Return type**
bool

**Returns**
True if there exists another job with the same key, False otherwise.

**Parameters**
**job** (*JobType*)

**downsize**()

class angr.analyses.**Identifier**

Bases: *Analysis*

**__init__**(*cfg=None*, *require_predecessors=True*, *only_find=None*)

**run**(*only_find=None*)

**can_call_same_name**(*addr*, *name*)

**get_func_info**(*func*)

**static constrain_all_zero**(*before_state*, *state*, *regs*)

**identify_func**(*function*)

**check_tests**(*cfg_func*, *match_func*)

**map_callsites**()

**do_trace**(*addr_trace*, *reverse_accesses*, *func_info*)

**get_call_args**(*func*, *callsite*)

**static get_reg_name**(*arch*, *reg_offset*)

**Parameters**

- **arch** – the architecture
- **reg_offset** – Tries to find the name of a register given the offset in the registers.

**Returns**
The register name

**find_stack_vars_x86**(*func*)

**static make_initial_state**(*project*, *stack_length*)

**Returns**
an initial state with a symbolic stack and good options for rop

**static make_symbolic_state**(*project*, *reg_list*, *stack_length=80*)
converts an input state into a state with symbolic registers :return: the symbolic state

class angr.analyses.**InitializationFinder**

Bases: *ForwardAnalysis*, *Analysis*

Finds possible initializations for global data sections and generate an overlay to be used in other analyses later on.

class angr.analyses.**LanguageDetector**

Bases: *Analysis*

Detect the original programming language and compiler used to build a binary.

Supports detection of C (gcc, clang, msvc), Rust, Go, and Swift through multiple heuristic layers: DWARF debug info, .comment sections, symbol patterns, section names, and linked library names.

Usage:

```
result = project.analyses.LanguageDetector()
print(result.language)     # "rust"
print(result.compiler)     # "rustc"
print(result.confidence)   # "high"
print(result.evidence)     # ["symbol: __rust_alloc", ...]
```

**__init__**()

**property language:** *str*

**property compiler:** *str | None*

**property compiler_version:** *str | None*

**property confidence:** *LanguageDetectionConfidenceLevel*

**property evidence:** *list[str]*

**class** angr.analyses.**LoopAnalysis**

Bases: *Analysis*

Analyze loop nodes in a structured C code representation and extract relevant information about the loop, including - Loop block addresses - Loop exits - Loop type - Loop condition - Max iterations - Fixed iterations

**__init__**(*cfunc*)

**Parameters**

**cfunc** (*CFunction*)

**class** angr.analyses.**LoopFinder**

Bases: *Analysis*

Extracts all the loops from all the functions in a binary.

**__init__**(*functions=None*, *normalize=True*)

**class** angr.analyses.**LoopUnroller**

Bases: *Analysis*

Unroll a loop in an AIL graph for a specified number of iterations.

**__init__**(*graph*, *loop_body*, *unroll_times*, *save_original*, *loop_body_incomplete=False*)

**Parameters**

- **graph** (*DiGraph*)
- **loop_body** (*set[tuple[int, int | None]]*)
- **unroll_times** (*int*)
- **save_original** (*bool*)
- **loop_body_incomplete** (*bool*)

**class** angr.analyses.**PackingDetector**

Bases: *Analysis*

This analysis detects if a binary is likely packed or not. We may extend it to identify which packer is in use in the future.

PACKED_MIN_BYTES = 256

PACKED_ENTROPY_MIN_THRESHOLD = 0.88

__init__(*cfg=None*, *region_size_threshold=32*)

**Parameters**

- **cfg** (CFGModel | *None*)
- **region_size_threshold** (*int*)

analyze()

class angr.analyses.**PatchFinderAnalysis**

Bases: *Analysis*

Looks for binary patches using some basic heuristics: - Looking for interleaved functions - Looking for unaligned functions

__init__()

atypical_alignments: list[*AtypicallyAlignedFunction*]

possibly_patched_out: list[*PatchedOutFunctionality*]

class angr.analyses.**Pathfinder**

Bases: *Analysis*

__init__(*start_state*, *goal_addr*, *cfg*, *cache_size=10000*)

**Parameters**

- **start_state** (SimState)
- **goal_addr** (*int*)
- **cfg** (CFGModel)

cache_state(*state*)

**Parameters**
**state** (SimState)

marker_to_state(*marker*)

**Return type**
*SimState* | None

**Parameters**
**marker** (SimStateMarker)

analyze()

**Return type**
bool

find_best_hypothesis_path()

**Return type**
tuple[int, ...]

**diagnose_unsat**(*state*)

**Parameters**
**state** (SimState)

**test_path**(*bbl_addr_trace*)

**Return type**
*TestPathReport*

**Parameters**
**bbl_addr_trace** (*tuple[int, ...]*)

**class** angr.analyses.**PropagatorAnalysis**

Bases: *ForwardAnalysis*, *Analysis*

PropagatorAnalysis implements copy propagation. It propagates values (either constant values or variables) and expressions inside a block or across a function.

PropagatorAnalysis only supports VEX. For AIL, please use SPropagator.

PropagatorAnalysis performs certain arithmetic operations between constants, including but are not limited to:

- addition
- subtraction
- multiplication
- division
- xor

It also performs the following memory operations:

- Loading values from a known address
- Writing values to a stack variable

**property prop_key:** tuple[str | None, str, int, bool, bool, bool]

Gets a key that represents the function and the "flavor" of the propagation result.

**property replacements**

**class** angr.analyses.**ProximityGraphAnalysis**

Bases: *Analysis*

Generate a proximity graph.

**__init__**(*func*, *cfg_model*, *xrefs*, *decompilation=None*, *expand_funcs=None*)

**Parameters**

- **func** (Function)
- **cfg_model** (CFGModel)
- **xrefs** (XRefManager)
- **decompilation** (Decompiler | *None*)
- **expand_funcs** (*set[int]* | *None*)

class angr.analyses.**ReachingDefinitionsAnalysis**

Bases: *ForwardAnalysis*[*ReachingDefinitionsState*, NodeType, object, object, object], *Analysis*

ReachingDefinitionsAnalysis is a text-book implementation of a static data-flow analysis that works on either a function or a block. It supports both VEX and AIL. By registering observers to observation points, users may use this analysis to generate use-def chains, def-use chains, and reaching definitions, and perform other traditional data-flow analyses such as liveness analysis.

- I've always wanted to find a better name for this analysis. Now I gave up and decided to live with this name for the foreseeable future (until a better name is proposed by someone else).
- Aliasing is definitely a problem, and I forgot how aliasing is resolved in this implementation. I'll leave this as a post-graduation TODO.
- Some more documentation and examples would be nice.

**__init__**(*subject*, *func_graph=None*, *max_iterations=30*, *track_tmps=False*, *track_consts=True*, *observation_points=None*, *init_state=None*, *init_context=None*, *state_initializer=None*, *cc=None*, *function_handler=None*, *observe_all=False*, *visited_blocks=None*, *dep_graph=True*, *observe_callback=None*, *canonical_size=8*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *interfunction_level=0*, *track_liveness=True*, *func_addr=None*, *element_limit=5*, *merge_into_tops=True*, *variable_map=None*)

**Parameters**

- **subject** (*Subject* | Block | *Block* | *Function* | str) – The subject of the analysis: a function, or a single basic block
- **func_graph** – Alternative graph for function.graph.
- **max_iterations** – The maximum number of iterations before the analysis is terminated.
- **track_tmps** – Whether or not temporary variables should be taken into consideration during the analysis.
- **observation_points** (Iterable[tuple[Literal['insn', 'node', 'stmt', 'exit'], int | tuple[int, int] | tuple[int, int, int], *ObservationPointType*]] | None) – A collection of tuples of ("node"|"insn", ins_addr, OP_TYPE) defining where reaching definitions should be copied and stored. OP_TYPE can be OP_BEFORE or OP_AFTER.
- **init_state** (*ReachingDefinitionsState* | None) – An optional initialization state. The analysis creates and works on a copy. Default to None: the analysis then initialize its own abstract state, based on the given <Subject>.
- **init_context** – If init_state is not given, this is used to initialize the context field of the initial state's CodeLocation. The only default-supported type which may go here is a tuple of integers, i.e. a callstack. Anything else requires a custom FunctionHandler.
- **cc** – Calling convention of the function.
- **function_handler** (*FunctionHandler* | None) – The function handler to update the analysis state and results on function calls.
- **observe_all** – Observe every statement, both before and after.
- **visited_blocks** – A set of previously visited blocks.
- **dep_graph** (*DepGraph* | bool | None) – An initial dependency graph to add the result of the analysis to. Set it to None to skip dependency graph generation.
- **canonical_size** – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.

- **dep_graph** – Set this to True to generate a dependency graph for the subject. It will be available as *result.dep_graph*.
- **interfunction_level** (int) – The number of functions we should recurse into. This parameter is only used if function_handler is not provided.
- **track_liveness** (bool) – Whether to track liveness information. This can consume sizeable amounts of RAM on large functions. (e.g. ~15GB for a function with 4k nodes)
- **merge_into_tops** (bool) – Merge known values into TOP if TOP is present. If True: {TOP} V {0xabc} = {TOP} If False: {TOP} V {0xabc} = {TOP, 0xabc}
- **state_initializer** (RDAStateInitializer | None)
- **func_addr** (int | None)
- **element_limit** (int)

**property observed_results:** dict[tuple[str, int, int], *LiveDefinitions*]

**property all_definitions**

**property all_uses**

**property one_result**

**property dep_graph:** *DepGraph*

**property visited_blocks**

**get_reaching_definitions_by_insn**(*ins_addr*, *op_type*)

**get_reaching_definitions_by_node**(*node_addr*, *op_type*)

**node_observe**(*node_addr*, *state*, *op_type*, *node_idx=None*)

**Parameters**

- **node_addr** (int) – Address of the node.
- **state** (*ReachingDefinitionsState*) – The analysis state.
- **op_type** (*ObservationPointType*) – Type of the observation point. Must be one of the following: OP_BEFORE, OP_AFTER.
- **node_idx** (int | None) – ID of the node. Used in AIL to differentiate blocks with the same address.

**Return type**
None

**insn_observe**(*insn_addr*, *stmt*, *block*, *state*, *op_type*)

**Parameters**

- **insn_addr** (int) – Address of the instruction.
- **stmt** (*Statement* | IRStmt) – The statement.
- **block** (*Block* | Block) – The current block.
- **state** (*ReachingDefinitionsState*) – The abstract analysis state.
- **op_type** (*ObservationPointType*) – Type of the observation point. Must be one of the following: OP_BEORE, OP_AFTER.

**Return type**
None

**stmt_observe**(*stmt_idx*, *stmt*, *block*, *state*, *op_type*)

**Parameters**

- **stmt_idx** (int)
- **stmt** (*Statement* | IRStmt)
- **block** (*Block* | Block)
- **state** (*ReachingDefinitionsState*)
- **op_type** (*ObservationPointType*)

**Return type**
None

**Returns**

**exit_observe**(*node_addr*, *exit_stmt_idx*, *block*, *state*, *node_idx=None*)

**Parameters**

- **node_addr** (*int*)
- **exit_stmt_idx** (*int*)
- **block** (Block | *Block*)
- **state** (ReachingDefinitionsState)
- **node_idx** (*int* | *None*)

**property subject**

**callsites_to**(*target*)

**Return type**
Iterable[*FunctionCallRelationships*]

**Parameters**
**target** (*int* | *str* | Function)

**class** angr.analyses.**Reassembler**

Bases: *Analysis*

High-level representation of a binary with a linear representation of all instructions and data regions. After calling "symbolize", it essentially acts as a binary reassembler.

Tested on CGC, x86 and x86-64 binaries.

Disclaimer: The reassembler is an empirical solution. Don't be surprised if it does not work on some binaries.

**__init__**(*syntax='intel'*, *remove_cgc_attachments=True*, *log_relocations=True*)

**property instructions**

Get a list of all instructions in the binary

**Returns**
A list of (address, instruction)

**Return type**
tuple

property relocations

property inserted_asm_before_label

property inserted_asm_after_label

property main_executable_regions

return:

property main_nonexecutable_regions

return:

disassemble_block(*block*)

Disassemble a block using the assembly syntax configured for this Reassembler.

When AT&T syntax is requested, a capstone instance dedicated to this analysis is used, so that neither the project-wide Arch object nor the capstone instance cached on it is modified.

**Parameters**
**block** – The angr Block to disassemble.

**Returns**
A CapstoneBlock instance.

**Return type**
*CapstoneBlock*

section_alignment(*section_name*)

Get the alignment for the specific section. If the section is not found, 16 is used as default.

**Parameters**
**section_name** (*str*) – The section.

**Returns**
The alignment in bytes.

**Return type**
int

main_executable_regions_contain(*addr*)

**Parameters**
**addr**

**Returns**

main_executable_region_limbos_contain(*addr*)

Sometimes there exists a pointer that points to a few bytes before the beginning of a section, or a few bytes after the beginning of the section. We take care of that here.

**Parameters**
**addr** (*int*) – The address to check.

**Returns**
A 2-tuple of (bool, the closest base address)

**Return type**
tuple

main_nonexecutable_regions_contain(*addr*)

**Parameters**
**addr** (*int*) – The address to check.

**Returns**
True if the address is inside a non-executable region, False otherwise.

**Return type**
bool

**main_nonexecutable_region_limbos_contain**(*addr*, *tolerance_before=64*, *tolerance_after=64*)

Sometimes there exists a pointer that points to a few bytes before the beginning of a section, or a few bytes after the beginning of the section. We take care of that here.

**Parameters**
**addr** (*int*) – The address to check.

**Returns**
A 2-tuple of (bool, the closest base address)

**Return type**
tuple

**register_instruction_reference**(*insn_addr*, *ref_addr*, *sort*, *operand_offset*)

**register_data_reference**(*data_addr*, *ref_addr*)

**add_label**(*name*, *addr*)

Add a new label to the symbol manager.

**Parameters**

- **name** (*str*) – Name of the label.
- **addr** (*int*) – Address of the label.

**Returns**
None

**insert_asm**(*addr*, *asm_code*, *before_label=False*)

Insert some assembly code at the specific address. There must be an instruction starting at that address.

**Parameters**

- **addr** (*int*) – Address of insertion
- **asm_code** (*str*) – The assembly code to insert

**Returns**
None

**append_procedure**(*name*, *asm_code*)

Add a new procedure with specific name and assembly code.

**Parameters**

- **name** (*str*) – The name of the new procedure.
- **asm_code** (*str*) – The assembly code of the procedure

**Returns**
None

**append_data**(*name*, *initial_content*, *size*, *readonly=False*, *sort='unknown'*)

Append a new data entry into the binary with specific name, content, and size.

**Parameters**

- **name** (*str*) – Name of the data entry. Will be used as the label.

- **initial_content** (bytes) – The initial content of the data entry.
- **size** (*int*) – Size of the data entry.
- **readonly** (*bool*) – If the data entry belongs to the readonly region.
- **sort** (*str*) – Type of the data.

**Returns**
None

**remove_instruction**(*ins_addr*)

**Parameters**
**ins_addr**

**Returns**

**randomize_procedures**()

**Returns**

**symbolize**()

**assembly**(*comments=False*, *symbolized=True*)

**remove_cgc_attachments**()
Remove CGC attachments.

**Returns**
True if CGC attachments are found and removed, False otherwise

**Return type**
bool

**remove_unnecessary_stuff**()
Remove unnecessary functions and data

**Returns**
None

**remove_unnecessary_stuff_glibc**()

**fast_memory_load**(*addr*, *size*, *data_type*, *endness='Iend_LE'*)
Load memory bytes from loader's memory backend.

**Parameters**

- **addr** (*int*) – The address to begin memory loading.
- **size** (*int*) – Size in bytes.
- **data_type** – Type of the data.
- **endness** (*str*) – Endianness of this memory load.

**Returns**
Data read out of the memory.

**Return type**
int or *bytes* or str or None

class angr.analyses.SLivenessAnalysis

Bases: *Analysis*

Calculates LiveIn and LiveOut sets for each block in a partial-SSA function.

__init__(*func*, *func_graph*, *entry=None*, *func_addr=None*, *arg_vvars=None*)

interference_graph()

Generate an interference graph based on the liveness analysis result.

**Returns**
A networkx.Graph instance.

**Return type**
networkx.Graph[int]

live_vars_by_stmt()

Get a mapping from statements to live variables at the point of the statement.

**Return type**
defaultdict[tuple[int, int | None], dict[int, set[int]]]

**Returns**
A dictionary mapping statements to sets of live variable IDs.

class angr.analyses.SPropagator

Bases: object

Constant and expression propagation that only supports SSA AIL graphs.

Deliberately not an Analysis for better speed.

__init__(*project*, *subject*, ***, *ail_manager*, *func_graph=None*, *only_consts=True*, *stack_pointer_tracker=None*, *func_args=None*, *func_addr=None*, *stack_arg_offsets=None*)

**Parameters**

- **project** (*Project*)
- **subject** (*Block | Function*)
- **ail_manager** (*Manager*)
- **func_graph** (*DiGraph | None*)
- **only_consts** (*bool*)
- **func_args** (*set[VirtualVariable] | None*)
- **func_addr** (*int | None*)
- **stack_arg_offsets** (*set[int] | None*)

property replacements

property dead_vvar_ids

static is_global_variable_updated(*func_graph*, *block_dict*, *varid*, *gv_addr*, *gv_size*, *defloc*, *useloc*)

**Return type**
bool

**Parameters**

- **varid** (*int*)

- **gv_addr** (*int*)
- **gv_size** (*int*)
- **defloc** (AILCodeLocation)
- **useloc** (AILCodeLocation)

**static is_vvar_used_for_addr_loading_switch_case**(*uselocs*, *blocks*)

Check if a virtual variable is used for loading an address in a switch-case construct.

**Parameters**

- **uselocs** (set[*AILCodeLocation*]) – The use locations of the virtual variable.
- **blocks** – All blocks of the current function.

**Return type**
bool

**Returns**
True if the virtual variable is used for loading an address in a switch-case construct, False otherwise.

**static replace**(*replacements*, *loc*, *expr*, *value*)

**Return type**
None

**Parameters**

- **replacements** (*dict[*AILCodeLocation, *dict[*VirtualVariable | Tmp, Expression*]]*)
- **loc** (AILCodeLocation)
- **expr** (VirtualVariable | Tmp)
- **value** (Expression)

**class** angr.analyses.**SPropagatorAnalysis**

Bases: *Analysis*, *SPropagator*

A wrapper around SPropagator to make it an Analysis.

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.**SReachingDefinitions**

Bases: object

Constant and expression propagation that only supports SSA AIL graphs.

Deliberately not an Analysis: it is instantiated hundreds of times per decompilation (often once per block), so it skips the analysis-factory ceremony. Instantiate it directly with the project as the first argument; exceptions always propagate.

**__init__**(*project*, *subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*, *variable_map=None*)

**class** angr.analyses.**SReachingDefinitionsAnalysis**

Bases: *Analysis*, *SReachingDefinitions*

A wrapper around SReachingDefinitions to make it usable as an Analysis and registered in the analysis hub.

__init__(*subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*, *variable_map=None*)

class angr.analyses.**SelfModifyingCodeAnalysis**

Bases: *Analysis*

Determine if some piece of code is self-modifying.

This determination is made by simply executing. If an address is executed that is also written to, the code is determined to be self-modifying. The determination is stored in the *result* property. The *regions* property contains a list of (addr, length) regions that were both written to and executed.

__init__(*subject*, *max_bytes=0*, *state=None*)

**Parameters**

- **subject** (int | str | *Function* | None) – Subject of analysis
- **max_bytes** (int) – Maximum number of bytes from subject address. 0 for no limit (default).
- **state** (*SimState* | None) – State to begin executing from.

**regions:** list[tuple[int, int]]

**result:** bool

class angr.analyses.**SootClassHierarchy**

Bases: *Analysis*

Generate complete hierarchy.

__init__()

**init_hierarchy**()

**has_super_class**(*cls*)

**is_subclass_including**(*cls_child*, *cls_parent*)

**is_subclass**(*cls_child*, *cls_parent*)

**is_visible_method**(*cls*, *method*)

**is_visible_class**(*cls_from*, *cls_to*)

**get_super_classes**(*cls*)

**get_super_classes_including**(*cls*)

**get_implementers**(*interface*)

**get_sub_interfaces_including**(*interface*)

**get_sub_interfaces**(*interface*)

**get_sub_classes**(*cls*)

**get_sub_classes_including**(*cls*)

**resolve_abstract_dispatch**(*cls*, *method*)

**resolve_concrete_dispatch**(*cls*, *method*)

**resolve_special_dispatch**(*method*, *container*)

**resolve_invoke**(*invoke_expr*, *method*, *container*)

**class** angr.analyses.**StackPointerTracker**

Bases: *Analysis*, *ForwardAnalysis*

Track the offset of stack pointer at the end of each basic block of a function.

**offset_after**(*addr*, *reg*)

**offset_before**(*addr*, *reg*)

**offset_after_block**(*block_addr*, *reg*)

**offset_before_block**(*block_addr*, *reg*)

**constant_after**(*addr*, *reg*)

**constant_before**(*addr*, *reg*)

**constant_after_block**(*block_addr*, *reg*)

**constant_before_block**(*block_addr*, *reg*)

**property inconsistent**

**inconsistent_for**(*reg*)

**offsets_for**(*reg*)

**class** angr.analyses.**StaticHooker**

Bases: *Analysis*

This analysis works on statically linked binaries - it finds the library functions statically linked into the binary and hooks them with the appropriate simprocedures.

Right now it only works on unstripped binaries, but hey! There's room to grow!

**__init__**(*library*, *binary=None*)

**class** angr.analyses.**StaticObjectFinder**

Bases: *Analysis*

**This analysis tries to find objects on the heap based on calls to new(), and subsequent calls to constructors with**
the 'this' pointer

**__init__**()

**class** angr.analyses.**Typehoon**

Bases: *Analysis*

A spiritual tribute to the long-standing typehoon project that @jmg (John Grosen) worked on during his days in the angr team. Now I feel really bad of asking the poor guy to work directly on VEX IR without any fancy static analysis support as we have right now...

Typehoon analysis implements a pushdown system that simplifies and solves type constraints. Our type constraints are largely an implementation of the paper Polymorphic Type Inference for Machine Code by Noonan, Loginov, and Cok from GrammaTech (with missing functionality support and bugs, of course). Type constraints

are collected by running VariableRecoveryFast (maybe VariableRecovery later as well) on a function, and then solved using this analysis.

User may specify ground truth, which will override all types at certain program points during constraint solving.

**__init__**(*constraints*, *func_var*, *ground_truth=None*, *var_mapping=None*, *must_struct=None*, *stackvar_max_sizes=None*, *stack_offset_tvs=None*, *constraint_set_degradation_threshold=150*, *type_translator=None*, *tv_manager=None*)

**Parameters**

- **constraints**
- **ground_truth** – A set of SimType-style solutions for some or all type variables. They will be respected during type solving.
- **var_mapping** (dict[*SimVariable*, set[*TypeVariable*]] | None)
- **must_struct** (set[*TypeVariable*] | None)
- **stackvar_max_sizes** (*dict[TypeVariable, int] | None*)
- **stack_offset_tvs** (*dict[int, TypeVariable] | None*)
- **constraint_set_degradation_threshold** (*int*)
- **type_translator** (*TypeTranslator | None*)
- **tv_manager** (*TypeVariableManager | None*)

**update_variable_types**(*func_addr*, *var_to_typevars*, *stack_offset_tvs=None*)

**Return type**
None

**Parameters**

- **func_addr** (*int | str*)
- **var_to_typevars** (*dict[SimVariable, set[TypeVariable]]*)
- **stack_offset_tvs** (*dict[int, TypeVariable] | None*)

**pp_constraints**()

Pretty-print constraints between *variables* using the variable mapping.

**Return type**
None

**pp_solution**()

Pretty-print solutions using the variable mapping.

**Return type**
None

**class** angr.analyses.**VariableRecovery**

Bases: *ForwardAnalysis*, *VariableRecoveryBase*

Recover "variables" from a function using forced execution.

While variables play a very important role in programming, it does not really exist after compiling. However, we can still identify and recovery their counterparts in binaries. It is worth noting that not every variable in source code can be identified in binaries, and not every recognized variable in binaries have a corresponding variable in the original source code. In short, there is no guarantee that the variables we identified/recognized in a binary are the same variables in its source code.

This analysis uses heuristics to identify and recovers the following types of variables: - Register variables. - Stack variables. - Heap variables. (not implemented yet) - Global variables. (not implemented yet)

This analysis takes a function as input, and performs a data-flow analysis on nodes. It runs concrete execution on every statement and hooks all register/memory accesses to discover all places that are accessing variables. It is slow, but has a more accurate analysis result. For a fast but inaccurate variable recovery, you may consider using VariableRecoveryFast.

This analysis follows SSA, which means every write creates a new variable in registers or memory (statck, heap, etc.). Things may get tricky when overlapping variable (in memory, as you cannot really have overlapping accesses to registers) accesses exist, and in such cases, a new variable will be created, and this new variable will overlap with one or more existing variables. A decision procedure (which is pretty much TODO) is required at the end of this analysis to resolve the conflicts between overlapping variables.

**__init__**(*func*, *max_iterations=20*, *store_live_variables=False*)

**Parameters**
**func** (*knowledge.Function*) – The function to analyze.

**class** angr.analyses.**VariableRecoveryFast**

Bases: *ForwardAnalysis*, *VariableRecoveryBase*

Recover "variables" from a function by keeping track of stack pointer offsets and pattern matching VEX statements.

If calling conventions are recovered prior to running VariableRecoveryFast, variables can be recognized more accurately. However, it is not a requirement. In this case, the function graph you pass must contain information indicating the call-out sites inside the analyzed function. These graph edges must be annotated with either `"type": "call"` or `"outside": True`.

**class** angr.analyses.**Veritesting**

Bases: *Analysis*

An exploration technique made for condensing chunks of code to single (nested) if-then-else constraints via CFG accurate to conduct Static Symbolic Execution SSE (conversion to single constraint)

**cfg_cache = {}**

**all_stashes = ('successful', 'errored', 'deadended', 'deviated', 'unconstrained')**

**__init__**(*input_state*, *boundaries=None*, *loop_unrolling_limit=10*, *enable_function_inlining=False*, *terminator=None*, *deviation_filter=None*)

SSE stands for Static Symbolic Execution, and we also implemented an extended version of Veritesting (Avgerinos, Thanassis, et al, ICSE 2014).

**Parameters**

- **input_state** – The initial state to begin the execution with.
- **boundaries** – Addresses where execution should stop.
- **loop_unrolling_limit** – The maximum times that Veritesting should unroll a loop for.
- **enable_function_inlining** – Whether we should enable function inlining and syscall inlining.
- **terminator** – A callback function that takes a state as parameter. Veritesting will terminate if this function returns True.
- **deviation_filter** – A callback function that takes a state as parameter. Veritesting will put the state into "deviated" stash if this function returns True.

**is_not_in_cfg**(*s*)

Returns if s.addr is not a proper node in our CFG.

**Parameters**
**s** (SimState) – The SimState instance to test.

**Returns bool**
False if our CFG contains p.addr, True otherwise.

**is_overbound**(*state*)

Filter out all states that run out of boundaries or loop too many times.

param SimState state: SimState instance to check returns bool: True if outside of mem/loop_ctr boundary

**class** angr.analyses.**VtableFinder**

Bases: *Analysis*

This analysis locates Vtables in a binary based on heuristics taken from - "Reconstruction of Class Hierarchies for Decompilation of C++ Programs"

**__init__**()

**is_cross_referenced**(*addr*)

**is_function**(*addr*)

**analyze**()

**create_extract_vtable**(*start_addr*, *sec_size*)

**class** angr.analyses.**XRefsAnalysis**

Bases: *ForwardAnalysis*, *Analysis*

XRefsAnalysis recovers in-depth x-refs (cross-references) in disassembly code.

Here is an example:

```
.text:
000023C8                LDR     R2, =time_now
000023CA                LDR     R3, [R2]
000023CC                ADDS    R3, #1
000023CE                STR     R3, [R2]
000023D0                BX      LR

.bss:
1FFF36F4 time_now       % 4
```

You will have the following x-refs for time_now:

```
23c8 - offset
23ca - read access
23ce - write access
```

angr.analyses.**register_analysis**(*cls*, *name*)

### Submodules

*analysis*
*backward_slice*
*binary_optimizer*
*bindiff*
*boyscout*
*callee_cleanup_finder*
*calling_convention*
*cdg*
*cfg*
*cfg_slice_to_sink*
*class_identifier*
*code_tagging*
*codecave*
*complete_calling_conventions*
*congruency_check*
*data_dep*
*datagraph_meta*
*ddg*
*decompiler*
*deobfuscator*
*disassembly*
*disassembly_utils*
*dominance_frontier*
*fcp*
*find_objects_static*
*flirt*
*forward_analysis*
*identifier*
*init_finder*
*language_detector*
*loop_analysis*
*loop_unroller*
*loopfinder*
*outliner*
*patchfinder*
*pathfinder*
*propagator*
*proximity_graph*
*purity*
*reaching_definitions*
*reassembler*
*s_liveness*
*s_propagator*
*s_reaching_definitions*
*smc*
*soot_class_hierarchy*
*stack_pointer_tracker*
*static_hooker*
*typehoon*
*unpacker*

continues on next page

Table 3 – continued from previous page

| |
|---|
| *variable_recovery* |
| *veritesting* |
| *vfg* |
| *vsa_ddg* |
| *vtable* |
| *xrefs* |

### 10.3.1 angr.analyses.analysis

**class** angr.analyses.analysis.**AnalysisLogEntry**

Bases: object

**__init__**(*message*, *exc_info=False*)

**format**()

**Return type**
str

**class** angr.analyses.analysis.**AnalysesHub**

Bases: *PluginVendor*[Any]

This class contains functions for all the registered and runnable analyses,

**__init__**(*project*)

**class** angr.analyses.analysis.**KnownAnalysesPlugin**

Bases: Protocol

**Identifier:** *type*[*Identifier*]

**CalleeCleanupFinder:** *type*[*CalleeCleanupFinder*]

**VSA_DDG:** *type*[*VSA_DDG*]

**CDG:** *type*[*CDG*]

**BinDiff:** *type*[*BinDiff*]

**CFGEmulated:** *type*[*CFGEmulated*]

**CFB:** *type*[*CFBlanket*]

**CFBlanket:** *type*[*CFBlanket*]

**CFG:** *type*[*CFG*]

**CFGFast:** *type*[*CFGFast*]

**StaticHooker:** *type*[*StaticHooker*]

**DDG:** *type*[*DDG*]

**CongruencyCheck:** *type*[*CongruencyCheck*]

**Reassembler:** *type*[*Reassembler*]

**BackwardSlice:** *type*[*BackwardSlice*]

**BinaryOptimizer:** *type*[*BinaryOptimizer*]

**VFG:** *type*[*VFG*]

**LoopFinder:** *type*[*LoopFinder*]

**Disassembly:** *type*[*Disassembly*]

**Veritesting:** *type*[*Veritesting*]

**CodeTagging:** *type*[*CodeTagging*]

**BoyScout:** *type*[*BoyScout*]

**VariableRecoveryFast:** *type*[*VariableRecoveryFast*]

**VariableRecovery:** *type*[*VariableRecovery*]

**ReachingDefinitions:** *type*[*ReachingDefinitionsAnalysis*]

**CompleteCallingConventions:** *type*[*CompleteCallingConventionsAnalysis*]

**Clinic:** *type*[*Clinic*]

**Propagator:** *type*[*PropagatorAnalysis*]

**CallingConvention:** *type*[*CallingConventionAnalysis*]

**Decompiler:** *type*[*Decompiler*]

**XRefs:** *type*[*XRefsAnalysis*]

**__init__**(**args*, ***kwargs*)

class angr.analyses.analysis.**AnalysesHubWithDefault**

Bases: *AnalysesHub*, *KnownAnalysesPlugin*

This class has type-hinting for all built-in analyses plugin

class angr.analyses.analysis.**AnalysisFactory**

Bases: Generic

**__init__**(*project*, *analysis_cls*)

**Parameters**

- **project** (Project)
- **analysis_cls** (type*[A]*)

**prep**(*fail_fast=None*, *kb=None*, *progress_callback=None*, *show_progressbar=False*)

**Return type**
type[TypeVar(A, bound= *Analysis*)]

**Parameters**

- **fail_fast** (*bool* | *None*)
- **kb** (KnowledgeBase | *None*)
- **progress_callback** (Callable | *None*)
- **show_progressbar** (*bool*)

class angr.analyses.analysis.**Analysis**

Bases: object

This class represents an analysis on the program.

**Variables**

- **project** – The project for this analysis.
- **kb** (KnowledgeBase) – The knowledgebase object.
- **_progress_callback** – A callback function for receiving the progress of this analysis. It only takes one argument, which is a float number from 0.0 to 100.0 indicating the current progress.
- **_show_progressbar** (*bool*) – If a progressbar should be shown during the analysis. It's independent from _progress_callback.
- **_progressbar** (*progress.Progress*) – The progress bar object.

**project:** *Project*

**kb:** *KnowledgeBase*

**errors:** list[*AnalysisLogEntry*] = []

**named_errors:** defaultdict[str, list[*AnalysisLogEntry*]] = {}

**log:** list

**property ram_usage:** float

Return the current RAM usage of the Python process, in bytes. The value is updated at most once per second.

angr.analyses.analysis.**register_analysis**(*cls*, *name*)

### 10.3.2 angr.analyses.backward_slice

class angr.analyses.backward_slice.**BackwardSlice**

Bases: *Analysis*

Represents a backward slice of the program.

**__init__**(*cfg*, *cdg*, *ddg*, *targets=None*, *cfg_node=None*, *stmt_id=None*, *control_flow_slice=False*, *same_function=False*, *no_construct=False*)

Create a backward slice from a specific statement based on provided control flow graph (CFG), control dependence graph (CDG), and data dependence graph (DDG).

The data dependence graph can be either CFG-based, or Value-set analysis based. A CFG-based DDG is much faster to generate, but it only reflects those states while generating the CFG, and it is neither sound nor accurate. The VSA based DDG (called VSA_DDG) is based on static analysis, which gives you a much better result.

**Parameters**

- **cfg** – The control flow graph.
- **cdg** – The control dependence graph.
- **ddg** – The data dependence graph.
- **targets** – A list of "target" that specify targets of the backward slices. Each target can be a tuple in form of (cfg_node, stmt_idx), or a CodeLocation instance.

- **cfg_node** – Deprecated. The target CFGNode to reach. It should exist in the CFG.
- **stmt_id** – Deprecated. The target statement to reach.
- **control_flow_slice** – True/False, indicates whether we should slice only based on CFG. Sometimes when acquiring DDG is difficult or impossible, you can just create a slice on your CFG. Well, if you don't even have a CFG, then…
- **no_construct** – Only used for testing and debugging to easily create a BackwardSlice object.

**dbg_repr**(*max_display=10*)

Debugging output of this slice.

**Parameters**
**max_display** – The maximum number of SimRun slices to show.

**Returns**
A string representation.

**dbg_repr_run**(*run_addr*)

Debugging output of a single SimRun slice.

**Parameters**
**run_addr** – Address of the SimRun.

**Returns**
A string representation.

**annotated_cfg**(*start_point=None*)

Returns an AnnotatedCFG based on slicing result.

**is_taint_related_to_ip**(*simrun_addr*, *stmt_idx*, *taint_type*, *simrun_whitelist=None*)

Query in taint graph to check if a specific taint will taint the IP in the future or not. The taint is specified with the tuple (simrun_addr, stmt_idx, taint_type).

**Parameters**

- **simrun_addr** – Address of the SimRun.
- **stmt_idx** – Statement ID.
- **taint_type** – Type of the taint, might be one of the following: 'reg', 'tmp', 'mem'.
- **simrun_whitelist** – A list of SimRun addresses that are whitelisted, i.e. the tainted exit will be ignored if it is in those SimRuns.

**Returns**
True/False

**is_taint_impacting_stack_pointers**(*simrun_addr*, *stmt_idx*, *taint_type*, *simrun_whitelist=None*)

Query in taint graph to check if a specific taint will taint the stack pointer in the future or not. The taint is specified with the tuple (simrun_addr, stmt_idx, taint_type).

**Parameters**

- **simrun_addr** – Address of the SimRun.
- **stmt_idx** – Statement ID.
- **taint_type** – Type of the taint, might be one of the following: 'reg', 'tmp', 'mem'.
- **simrun_whitelist** – A list of SimRun addresses that are whitelisted.

**Returns**
True/False.

### 10.3.3 angr.analyses.binary_optimizer

**class** angr.analyses.binary_optimizer.**ConstantPropagation**

Bases: object

**__init__**(*constant*, *constant_assignment_loc*, *constant_consuming_loc*)

**class** angr.analyses.binary_optimizer.**RedundantStackVariable**

Bases: object

**__init__**(*argument*, *stack_variable*, *stack_variable_consuming_locs*)

**class** angr.analyses.binary_optimizer.**RegisterReallocation**

Bases: object

**__init__**(*stack_variable*, *register_variable*, *stack_variable_sources*, *stack_variable_consumers*, *prologue_addr*, *prologue_size*, *epilogue_addr*, *epilogue_size*)

Constructor.

**Parameters**

- **stack_variable** (SimStackVariable)
- **register_variable** (SimRegisterVariable)
- **stack_variable_sources** (*list*)
- **stack_variable_consumers** (*list*)
- **prologue_addr** (*int*)
- **prologue_size** (*int*)
- **epilogue_addr** (*int*)
- **epilogue_size** (*int*)

**class** angr.analyses.binary_optimizer.**DeadAssignment**

Bases: object

**__init__**(*pv*)

Constructor.

**Parameters**
**pv** (angr.analyses.ddg.ProgramVariable) – The assignment to remove.

**class** angr.analyses.binary_optimizer.**BinaryOptimizer**

Bases: *Analysis*

This is a collection of binary optimization techniques we used in Mechanical Phish during the finals of Cyber Grand Challenge. It focuses on dealing with some serious speed-impacting code constructs, and *sort of* worked on *some* CGC binaries compiled with O0. Use this analysis as a reference of how to use data dependency graph and such.

There is no guarantee that BinaryOptimizer will ever work on non-CGC binaries. Feel free to give us PR or MR, but please *do not* ask for support of non-CGC binaries.

**BLOCKS_THRESHOLD = 500**

__init__(*cfg*, *techniques*)

**Parameters**
**techniques** (*set[str]*)

optimize()

### 10.3.4 angr.analyses.bindiff

exception angr.analyses.bindiff.**UnmatchedStatementsException**

Bases: Exception

class angr.analyses.bindiff.**Difference**

Bases: object

__init__(*diff_type*, *value_a*, *value_b*)

class angr.analyses.bindiff.**ConstantChange**

Bases: object

__init__(*offset*, *value_a*, *value_b*)

angr.analyses.bindiff.**differing_constants**(*block_a*, *block_b*)

Compares two basic blocks and finds all the constants that differ from the first block to the second.

**Parameters**

- **block_a** – The first block to compare.
- **block_b** – The second block to compare.

**Returns**
Returns a list of differing constants in the form of ConstantChange, which has the offset in the block and the respective constants.

angr.analyses.bindiff.**compare_statement_dict**(*statement_1*, *statement_2*)

class angr.analyses.bindiff.**NormalizedBlock**

Bases: object

__init__(*block*, *function*)

class angr.analyses.bindiff.**NormalizedFunction**

Bases: object

__init__(*function*)

**Parameters**
**function** (Function)

class angr.analyses.bindiff.**FunctionDiff**

Bases: object

This class computes the a diff between two functions.

__init__(*function_a*, *function_b*, *bindiff=None*)

**Parameters**

- **function_a** (*Function*) – The first angr Function object to diff.
- **function_b** (*Function*) – The second angr Function object.

- **bindiff** – An optional Bindiff object. Used for some extra normalization during basic block comparison.

**property probably_identical**

Whether or not these two functions are identical.

**Type**
returns

**property identical_blocks**

A list of block matches which appear to be identical

**Type**
returns

**property differing_blocks**

A list of block matches which appear to differ

**Type**
returns

**property blocks_with_differing_constants**

A list of block matches which appear to differ

**Type**
return

**property block_matches**

**property unmatched_blocks**

**static get_normalized_block**(*addr*, *function*)

**Parameters**

- **addr** – Where to start the normalized block.
- **function** – A function containing the block address.

**Returns**
A normalized basic block.

**block_similarity**(*block_a*, *block_b*)

**Parameters**

- **block_a** – The first block address.
- **block_b** – The second block address.

**Returns**
The similarity of the basic blocks, normalized for the base address of the block and function call addresses.

**blocks_probably_identical**(*block_a*, *block_b*, *check_constants=False*)

**Parameters**

- **block_a** – The first block address.
- **block_b** – The second block address.
- **check_constants** – Whether or not to require matching constants in blocks.

**Returns**
Whether or not the blocks appear to be identical.

class angr.analyses.bindiff.**BinDiff**

Bases: *Analysis*

This class computes the a diff between two binaries represented by angr Projects

**__init__**(*other_project*, *cfg_a=None*, *cfg_b=None*)

**Parameters**
**other_project** – The second project to diff

**functions_probably_identical**(*func_a_addr*, *func_b_addr*, *check_consts=False*)

Compare two functions and return True if they appear identical.

**Parameters**

- **func_a_addr** – The address of the first function (in the first binary).
- **func_b_addr** – The address of the second function (in the second binary).

**Returns**
Whether or not the functions appear to be identical.

**property identical_functions**

A list of function matches that appear to be identical

**Type**
returns

**property differing_functions**

A list of function matches that appear to differ

**Type**
returns

**differing_functions_with_consts**()

**Returns**
A list of function matches that appear to differ including just by constants

**property differing_blocks**

A list of block matches that appear to differ

**Type**
returns

**property identical_blocks**

return A list of all block matches that appear to be identical

**property blocks_with_differing_constants**

A dict of block matches with differing constants to the tuple of constants

**Type**
return

**property unmatched_functions**

**get_function_diff**(*function_addr_a*, *function_addr_b*)

**Parameters**

- **function_addr_a** – The address of the first function (in the first binary)
- **function_addr_b** – The address of the second function (in the second binary)

**Returns**
the FunctionDiff of the two functions

### 10.3.5 angr.analyses.boyscout

**class** angr.analyses.boyscout.**BoyScout**

Bases: *Analysis*

Try to determine the architecture and endieness of a binary blob

**__init__**(*cookiesize=1*)

### 10.3.6 angr.analyses.callee_cleanup_finder

**class** angr.analyses.callee_cleanup_finder.**CalleeCleanupFinder**

Bases: *Analysis*

**__init__**(*starts=None*, *hook_all=False*)

**analyze**(*addr*)

### 10.3.7 angr.analyses.calling_convention

**class** angr.analyses.calling_convention.**CallingConventionAnalysis**

Bases: *Analysis*

Analyze the calling convention of a function and guess a probable prototype.

The calling convention of a function can be inferred at both its call sites and the function itself. At call sites, we consider all register and stack variables that are not alive after the function call as parameters to this function. In the function itself, we consider all register and stack variables that are read but without initialization as parameters. Then we synthesize the information from both locations and make a reasonable inference of calling convention of this function.

**Variables**

- **_function** – The function to recover calling convention for.
- **_variable_manager** – A handy accessor to the variable manager.
- **_cfg** – A reference of the CFGModel of the current binary. It is used to discover call sites of the current function in order to perform analysis at call sites.
- **analyze_callsites** – True if we should analyze all call sites of the current function to determine the calling convention and arguments. This can be time-consuming if there are many call sites to analyze.
- **cc** (*SimCC* | None) – The recovered calling convention for the function.
- **_collect_facts** – True if we should run FunctionFactCollector to collect input arguments and return value size. False if input arguments and return value size are provided by the user.

**__init__**(*func*, *cfg=None*, *analyze_callsites=False*, *caller_func_addr=None*, *callsite_block_addr=None*, *callsite_insn_addr=None*, *func_graph=None*, *input_args=None*, *retval_size=None*, *extra_pop=None*, *collect_facts=False*, *collect_facts_arg_uses=False*, *collect_facts_arg_passthru=False*)

**Parameters**

- **func** (Function | *int* | *str* | *None*)
- **cfg** (CFGModel | *None*)
- **analyze_callsites** (*bool*)
- **caller_func_addr** (*int* | *None*)
- **callsite_block_addr** (*int* | *None*)
- **callsite_insn_addr** (*int* | *None*)
- **func_graph** (*DiGraph* | *None*)
- **input_args** (*list[SimRegArg* | *SimStackArg]* | *None*)
- **retval_size** (*int* | *None*)
- **extra_pop** (*int* | *None*)
- **collect_facts** (*bool*)
- **collect_facts_arg_uses** (*bool*)
- **collect_facts_arg_passthru** (*bool*)

**is_va_start_amd64**(*func*)

**Return type**
tuple[bool, int | None]

**Parameters**
**func** (Function)

**class** angr.analyses.calling_convention.**FactCollector**

Bases: *Analysis*

An extremely fast analysis that extracts necessary facts of a function for CallingConventionAnalysis to make decision on the calling convention and prototype of a function.

**__init__**(*func*, *max_depth=100*, *track_arg_uses=False*, *track_arg_passthru=False*)

**Parameters**

- **func** (Function)
- **max_depth** (*int*)
- **track_arg_uses** (*bool*)
- **track_arg_passthru** (*bool*)

**callsites**: dict[int, tuple[*Function*, list[*FactData*]]]

**input_args**: list[*SimRegArg* | *SimStackArg*] | None

**unused_args**: list[*SimRegArg*]

**retval_size**: int | None

**retval_incidental:** bool

True when every endpoint's write to the return register looks like a leftover rather than a return value: the value written is consumed again in the same block (stored, tested, passed on) after the write. A void function whose last instruction happens to load into rax looks exactly like a getter from the outside; this is the one local hint that it is not, and CallingConventionAnalysis only lets call-site evidence demote a prototype to void when it is set.

**pointer_arg_derefs:** defaultdict[*FactData*, int]

**extra_pop:** int | None

### Submodules

*calling_convention*
*fact_collector*
*utils*

### angr.analyses.calling_convention.calling_convention

**class** angr.analyses.calling_convention.calling_convention.**CallSiteFact**

Bases: object

Store facts about each call site.

**__init__**(*return_value_used*)

**class** angr.analyses.calling_convention.calling_convention.**UpdateArgumentsOption**

Bases: object

Enums for controlling the argument updating behavior in _adjust_cc.

**DoNotUpdate = 0**

**AlwaysUpdate = 1**

**UpdateWhenCCHasNoArgs = 2**

**class** angr.analyses.calling_convention.calling_convention.**CallingConventionAnalysis**

Bases: *Analysis*

Analyze the calling convention of a function and guess a probable prototype.

The calling convention of a function can be inferred at both its call sites and the function itself. At call sites, we consider all register and stack variables that are not alive after the function call as parameters to this function. In the function itself, we consider all register and stack variables that are read but without initialization as parameters. Then we synthesize the information from both locations and make a reasonable inference of calling convention of this function.

**Variables**

- **_function** – The function to recover calling convention for.
- **_variable_manager** – A handy accessor to the variable manager.
- **_cfg** – A reference of the CFGModel of the current binary. It is used to discover call sites of the current function in order to perform analysis at call sites.

- **analyze_callsites** – True if we should analyze all call sites of the current function to determine the calling convention and arguments. This can be time-consuming if there are many call sites to analyze.
- **cc** (*SimCC* | None) – The recovered calling convention for the function.
- **_collect_facts** – True if we should run FunctionFactCollector to collect input arguments and return value size. False if input arguments and return value size are provided by the user.

**__init__**(*func*, *cfg=None*, *analyze_callsites=False*, *caller_func_addr=None*, *callsite_block_addr=None*, *callsite_insn_addr=None*, *func_graph=None*, *input_args=None*, *retval_size=None*, *extra_pop=None*, *collect_facts=False*, *collect_facts_arg_uses=False*, *collect_facts_arg_passthru=False*)

**Parameters**

- **func** (Function | *int* | *str* | *None*)
- **cfg** (CFGModel | *None*)
- **analyze_callsites** (*bool*)
- **caller_func_addr** (*int* | *None*)
- **callsite_block_addr** (*int* | *None*)
- **callsite_insn_addr** (*int* | *None*)
- **func_graph** (*DiGraph* | *None*)
- **input_args** (*list[SimRegArg* | *SimStackArg]* | *None*)
- **retval_size** (*int* | *None*)
- **extra_pop** (*int* | *None*)
- **collect_facts** (*bool*)
- **collect_facts_arg_uses** (*bool*)
- **collect_facts_arg_passthru** (*bool*)

**cc**: *SimCC* | None

**prototype_libname**: str | None

**proto_from_symbol**: bool

**prototype**: *SimTypeFunction* | None

**is_va_start_amd64**(*func*)

**Return type**
tuple[bool, int | None]

**Parameters**
**func** (Function)

### angr.analyses.calling_convention.fact_collector

**type** angr.analyses.calling_convention.fact_collector.**FactData** = tuple[int, int, int] | None

class angr.analyses.calling_convention.fact_collector.**FactCollectorState**

Bases: object

The abstract state for FactCollector.

**__init__**()

**tmps**: dict[int, *FactData*]

**simple_stack**: dict[int, *FactData*]

**simple_regs**: dict[int, *FactData*]

**ins_addr**

**callee_stored_regs**: dict[int, int]

**reg_reads**

**reg_reads_count**

**reg_writes**: set[int]

**stack_reads**

**stack_writes**: set[int]

**pointer_arg_derefs**: defaultdict[*FactData*, int]

**sp_value**

**bp_value**

**register_read**(*offset*, *size_in_bytes*)

**Parameters**

- **offset** (*int*)
- **size_in_bytes** (*int*)

**register_read_undo**(*offset*)

**Return type**
None

**Parameters**
**offset** (*int*)

**register_written**(*offset*, *size_in_bytes*)

**Parameters**

- **offset** (*int*)
- **size_in_bytes** (*int*)

**stack_read**(*offset*, *size_in_bytes*)

**Parameters**

- **offset** (*int*)
- **size_in_bytes** (*int*)

**stack_written**(*offset*, *size_int_bytes*)

**Parameters**

- **offset** (*int*)
- **size_int_bytes** (*int*)

**copy**(*with_tmps=True*)

**Return type**
*FactCollectorState*

**Parameters**
**with_tmps** (*bool*)

**class** angr.analyses.calling_convention.fact_collector.**SimEngineFactCollectorVEX**

Bases: *SimEngineNostmtVEX*[*FactCollectorState*, *FactData*, None], *SimEngineLight*[*FactCollectorState*, *FactData*, *Block*, None]

The engine for FactCollector.

**__init__**(*project*, *bp_as_gpr*, *track_arg_uses*, *seen_reg_uses*)

**Parameters**

- **bp_as_gpr** (*bool*)
- **track_arg_uses** (*bool*)
- **seen_reg_uses** (*defaultdict[int, int]*)

**class** angr.analyses.calling_convention.fact_collector.**FactCollector**

Bases: *Analysis*

An extremely fast analysis that extracts necessary facts of a function for CallingConventionAnalysis to make decision on the calling convention and prototype of a function.

**__init__**(*func*, *max_depth=100*, *track_arg_uses=False*, *track_arg_passthru=False*)

**Parameters**

- **func** (Function)
- **max_depth** (*int*)
- **track_arg_uses** (*bool*)
- **track_arg_passthru** (*bool*)

**callsites:** dict[int, tuple[*Function*, list[*FactData*]]]

**input_args:** list[*SimRegArg* | *SimStackArg*] | None

**unused_args:** list[*SimRegArg*]

**retval_size:** int | None

**retval_incidental:** bool

True when every endpoint's write to the return register looks like a leftover rather than a return value: the value written is consumed again in the same block (stored, tested, passed on) after the write. A void function whose last instruction happens to load into rax looks exactly like a getter from the outside; this is the one local hint that it is not, and CallingConventionAnalysis only lets call-site evidence demote a prototype to void when it is set.

**pointer_arg_derefs:** defaultdict[*FactData*, int]

**extra_pop:** int | None

### angr.analyses.calling_convention.utils

angr.analyses.calling_convention.utils.**is_sane_register_variable**(*arch*, *reg_offset*, *reg_size*, *def_cc=None*)

Filters all registers that are surly not members of function arguments. This can be seen as a workaround, since VariableRecoveryFast sometimes gives input variables of cc_ndep (which is a VEX-specific register) :-(

**Parameters**

- **reg_offset** (int) – The register offset.
- **reg_size** (int) – The register size.
- **arch** (*Arch*)
- **def_cc** (SimCC | type[SimCC] | *None*)

**Returns**
True if it is an acceptable function argument, False otherwise.

**Return type**
bool

angr.analyses.calling_convention.utils.**merge_overlapping_register_spans**(*arch*, *spans*)

Merge register reads that belong to the same base register into one span per base register.

The merged span is the smallest named register starting at the base register that covers every read (e.g., reads of `ch` and `cx` become `cx`). When no register of the covering size exists, the widest read is kept. Floating-point and vector registers are left alone: their sub-registers hold separate arguments (e.g., `s0` and `s1` in `d0` on ARM hard-float).

**Parameters**

- **arch** (Arch) – The architecture.
- **spans** (Iterable[tuple[int, int]]) – Register reads as (offset, size) tuples.

**Return type**
list[tuple[int, int]]

**Returns**
Merged (offset, size) tuples, sorted by offset.

## 10.3.8 angr.analyses.cdg

**class** angr.analyses.cdg.**CDG**

Bases: *Analysis*

Implements a control dependence graph.

**__init__**(*cfg*, *start=None*, *no_construct=False*)

Constructor.

**Parameters**

- **cfg** – The control flow graph upon which this control dependence graph will build
- **start** – The starting point to begin constructing the control dependence graph

- **no_construct** – Skip the construction step. Only used in unit-testing.

**property graph**

**get_post_dominators**()

Return the post-dom tree

**get_dependants**(*run*)

Return a list of nodes that are control dependent on the given node in the control dependence graph

**get_guardians**(*run*)

Return a list of nodes on whom the specific node is control dependent in the control dependence graph

### 10.3.9 angr.analyses.cfg

**class** angr.analyses.cfg.**CFG**

Bases: *CFGFast*

tl;dr: CFG is just a wrapper around CFGFast for compatibility issues. It will be fully replaced by CFGFast in future releases. Feel free to use CFG if you intend to use CFGFast. Please use CFGEmulated if you *have to* use the old, slow, dynamically-generated version of CFG.

For multiple historical reasons, angr's CFG is accurate but slow, which does not meet what most people expect. We developed CFGFast for light-speed CFG recovery, and renamed the old CFG class to CFGEmulated. For compatibility concerns, CFG was kept as an alias to CFGEmulated.

However, so many new users of angr would load up a binary and generate a CFG immediately after running "pip install angr", and draw the conclusion that "angr's CFG is so slow - angr must be unusable!" Therefore, we made the hard decision: CFG will be an alias to CFGFast, instead of CFGEmulated.

To ease the transition of your existing code and script, the following changes are made:

- A CFG class, which is a sub class of CFGFast, is created.
- You will see both a warning message printed out to stderr and an exception raised by angr if you are passing CFG any parameter that only CFGEmulated supports. This exception is not a sub class of AngrError, so you wouldn't capture it with your old code by mistake.
- In the near future, this wrapper class will be removed completely, and CFG will be a simple alias to CFGFast.

We expect most interfaces are the same between CFGFast and CFGEmulated. Apparently some functionalities (like context-sensitivity, and state keeping) only exist in CFGEmulated, which is when you want to use CFGEmulated instead.

**class** angr.analyses.cfg.**CFBlanket**

Bases: *Analysis*

A Control-Flow Blanket is a representation for storing all instructions, data entries, and bytes of a full program.

Region types: - section - segment - extern - tls - kernel

**__init__**(*exclude_region_types=None*, *on_object_added=None*, *on_object_removed=None*)

**Parameters**

- **on_object_added** (Callable[[int, Any], None] | None) – Callable with parameters (addr, obj) called after an object is added to the blanket.
- **on_object_removed** (Callable[[int, Any], None] | None) – Callable with parameters (addr, obj) called after an object is removed from the blanket.
- **exclude_region_types** (*set[str] | None*)

**property regions**

Return all memory regions.

**floor_addr**(*addr*)

**Return type**
int

**Parameters**
**addr** (*int*)

**floor_item**(*addr*)

**Return type**
tuple[int, Any]

**Parameters**
**addr** (*int*)

**floor_items**(*addr=None*, *reverse=False*)

**ceiling_addr**(*addr*)

**Return type**
int

**Parameters**
**addr** (*int*)

**ceiling_item**(*addr*)

**Return type**
tuple[int, Any]

**Parameters**
**addr** (*int*)

**ceiling_items**(*addr=None*, *reverse=False*, *include_first=True*)

**add_obj**(*addr*, *obj*)

Add an object *obj* to the blanket at the specified address *addr*, keeping the blanket non-overlapping: existing objects that overlap [addr, addr + obj.size) are removed or trimmed, and the trimmed-off remainders are re-inserted. Objects without an integer size are stored as zero-span entries and never cause trimming.

**remove_obj**(*addr*, *fill=True*)

Remove the object at *addr* from the blanket. When *fill* is set (the default), the removed object's span is re-filled with an Unknown region so that the blanket remains a total cover of the mapped address space. Removing a missing address is a no-op.

**Returns**
The removed object, or None if no object exists at *addr*.

**Parameters**
**fill** (*bool*)

**add_function**(*func*)

Add a function *func* and all blocks of this function to the blanket.

**dbg_repr**()

The debugging representation of this CFBlanket.

**Returns**
The debugging representation of this CFBlanket.

**Return type**
str

class angr.analyses.cfg.**CFGArchOptions**

Bases: object

Stores architecture-specific options and settings, as well as the detailed explanation of those options and settings.

Suppose *ao* is the CFGArchOptions object, and there is an option called *ret_jumpkind_heuristics*, you can access it by *ao.ret_jumpkind_heuristics* and set its value via *ao.ret_jumpkind_heuristics = True*

**Variables**

- **OPTIONS** (*dict*) – A dict of all default options for different architectures.
- **arch** (*archinfo.Arch*) – The architecture object.
- **_options** (*dict*) – Values of all CFG options that are specific to the current architecture.

```
OPTIONS = {'ARMCortexM': {'has_arm_code': (<class 'bool'>, False),
'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class
'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, False)}, 'ARMEL':
{'has_arm_code': (<class 'bool'>, True), 'pattern_match_ifuncs': (<class 'bool'>,
True), 'ret_jumpkind_heuristics': (<class 'bool'>, True),
'switch_mode_on_nodecode': (<class 'bool'>, True)}, 'ARMHF': {'has_arm_code':
(<class 'bool'>, True), 'pattern_match_ifuncs': (<class 'bool'>, True),
'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode':
(<class 'bool'>, True)}}
```

__init__(*arch*, ***options*)

Constructor.

**Parameters**

- **arch** (*archinfo.Arch*) – The architecture instance.
- **options** (*dict*) – Architecture-specific options, which will be used to initialize this object.

```
arch = None
```

class angr.analyses.cfg.**CFGBase**

Bases: *Analysis*

The base class for control flow graphs.

```
tag: str = None
```

```
addr_type: Literal['int', 'block_id', 'soot'] = None
```

__init__(*sort*, *context_sensitivity_level*, *normalize=False*, *binary=None*, *objects=None*, *regions=None*, *exclude_sparse_regions=True*, *skip_specific_regions=True*, *force_segment=False*, *base_state=None*, *resolve_indirect_jumps=True*, *indirect_jump_resolvers=None*, *indirect_jump_target_limit=100000*, *detect_tail_calls=False*, *low_priority=False*, *skip_unmapped_addrs=True*, *sp_tracking_track_memory=True*, *model=None*)

**Parameters**

- **sort** (*str*) – 'fast' or 'emulated'.

- **context_sensitivity_level** (*int*) – The level of context-sensitivity of this CFG (see documentation for further details). It ranges from 0 to infinity.
- **normalize** (*bool*) – Whether the CFG as well as all Function graphs should be normalized.
- **binary** (*cle.backends.Backend*) – The binary to recover CFG on. By default, the main binary is used.
- **objects** – A list of objects to recover the CFG on. By default, it will recover the CFG of all loaded objects.
- **regions** (*iterable*) – A list of tuples in the form of (start address, end address) describing memory regions that the CFG should cover.
- **force_segment** (*bool*) – Force CFGFast to rely on binary segments instead of sections.
- **base_state** (*angr.SimState*) – A state to use as a backer for all memory loads.
- **resolve_indirect_jumps** (*bool*) – Whether to try to resolve indirect jumps. This is necessary to resolve jump targets from jump tables, etc.
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **indirect_jump_target_limit** (*int*) – Maximum indirect jump targets to be recovered.
- **skip_unmapped_addrs** – Ignore all branches into unmapped regions. True by default. You may want to set it to False if you are analyzing manually patched binaries or malware samples.
- **detect_tail_calls** (*bool*) – Aggressive tail-call optimization detection. This option is only respected in make_functions().
- **sp_tracking_track_memory** (*bool*) – Whether or not to track memory writes if tracking the stack pointer. This increases the accuracy of stack pointer tracking, especially for architectures without a base pointer. Only used if detect_tail_calls is enabled.
- **model** (*None or CFGModel*) – The CFGModel instance to write to. A new CFGModel instance will be created and registered with the knowledge base if *model* is None.

**Returns**
None

**indirect_jumps:** `dict[int, IndirectJump]`

**property model:** *CFGModel*

Get the CFGModel instance. :return: The CFGModel instance that this analysis currently uses.

**property normalized**

**property context_sensitivity_level**

**property functions**

A reference to the FunctionManager in the current knowledge base.

**Returns**
FunctionManager with all functions

**Return type**
angr.knowledge_plugins.FunctionManager

property regions: list[tuple[int, int]]

The memory regions that this analysis covers. An empty list means it had nothing to scan.

**Returns**
A sorted list of (start address, end address) tuples.

make_copy(*copy_to*)

Copy self attributes to the new object.

**Parameters**
**copy_to** (CFGBase) – The target to copy to.

**Returns**
None

copy()

output()

generate_index()

Generate an index of all nodes in the graph in order to speed up get_any_node() with anyaddr=True.

**Returns**
None

get_loop_back_edges()

property graph: *SpillingCFG*

remove_edge(*block_from*, *block_to*)

is_thumb_addr(*addr*)

record_memory_data_addr(*addr*)

Record the address of a newly added memory data object.

**Return type**
None

**Parameters**
**addr** (*int*)

reset_memory_data_addrs()

Reset the set of addresses of newly added memory data objects.

**Return type**
None

normalize()

Normalize the CFG, making sure that there are no overlapping basic blocks.

Note that this method will not alter transition graphs of each function in self.kb.functions. You may call normalize() on each Function object to normalize their transition graphs.

**Returns**
None

mark_function_alignments()

Find all potential function alignments and mark them.

Note that it is not always correct to simply remove them, because these functions may not be actual alignments but part of an actual function, and is incorrectly marked as an individual function because of failures

in resolving indirect jumps. An example is in the test binary `x86_64/dir_gcc_-O0` 0x40541d (indirect jump at 0x4051b0). If the indirect jump cannot be correctly resolved, removing function 0x40541d will cause a missing label failure in reassembler.

**Returns**
None

`make_functions()`

Revisit the entire control flow graph, create Function instances accordingly, and correctly put blocks into each function.

Although Function objects are created during the CFG recovery, they are neither sound nor accurate. With a pre-constructed CFG, this method rebuilds all functions bearing the following rules:

- A block may only belong to one function.
- Small functions lying inside the startpoint and the endpoint of another function will be merged with the other function
- Tail call optimizations are detected.
- PLT stubs are aligned by 16.

**Returns**
None

`class angr.analyses.cfg.CFGEmulated`

Bases: *ForwardAnalysis*, *CFGBase*

This class represents a control-flow graph.

`tag: str = 'CFGEmulated'`

`addr_type: Literal['int', 'block_id', 'soot'] = 'block_id'`

`__init__`(*context_sensitivity_level=1*, *start=None*, *avoid_runs=None*, *enable_function_hints=False*, *call_depth=None*, *call_tracing_filter=None*, *initial_state=None*, *starts=None*, *keep_state=False*, *indirect_jump_target_limit=100000*, *resolve_indirect_jumps=True*, *enable_advanced_backward_slicing=False*, *enable_symbolic_back_traversal=False*, *indirect_jump_resolvers=None*, *additional_edges=None*, *no_construct=False*, *normalize=False*, *max_iterations=1*, *address_whitelist=None*, *base_graph=None*, *iropt_level=None*, *max_steps=None*, *state_add_options=None*, *state_remove_options=None*, *model=None*)

All parameters are optional.

**Parameters**

- **context_sensitivity_level** – The level of context-sensitivity of this CFG (see documentation for further details). It ranges from 0 to infinity. Default 1.
- **avoid_runs** – A list of runs to avoid.
- **enable_function_hints** – Whether to use function hints (constants that might be used as exit targets) or not.
- **call_depth** – How deep in the call stack to trace.
- **call_tracing_filter** – Filter to apply on a given path and jumpkind to determine if it should be skipped when call_depth is reached.
- **initial_state** – An initial state to use to begin analysis.

- **starts** (*iterable*) – A collection of starting points to begin analysis. It can contain the following three different types of entries: an address specified as an integer, a 2-tuple that includes an integer address and a jumpkind, or a SimState instance. Unsupported entries in starts will lead to an AngrCFGError being raised.
- **keep_state** – Whether to keep the SimStates for each CFGNode.
- **resolve_indirect_jumps** – Whether to enable the indirect jump resolvers for resolving indirect jumps
- **enable_advanced_backward_slicing** – Whether to enable an intensive technique for resolving indirect jumps
- **enable_symbolic_back_traversal** – Whether to enable an intensive technique for resolving indirect jumps
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **additional_edges** – A dict mapping addresses of basic blocks to addresses of successors to manually include and analyze forward from.
- **no_construct** (*bool*) – Skip the construction procedure. Only used in unit-testing.
- **normalize** (*bool*) – If the CFG as well as all Function graphs should be normalized or not.
- **max_iterations** (*int*) – The maximum number of iterations that each basic block should be "executed". 1 by default. Larger numbers of iterations are usually required for complex analyses like loop analysis.
- **address_whitelist** (*iterable*) – A list of allowed addresses. Any basic blocks outside of this collection of addresses will be ignored.
- **base_graph** (*networkx.DiGraph*) – A basic control flow graph to follow. Each node inside this graph must have the following properties: *addr* and *size*. CFG recovery will strictly follow nodes and edges shown in the graph, and discard any control flow that does not follow an existing edge in the base graph. For example, you can pass in a Function local transition graph as the base graph, and CFGEmulated will traverse nodes and edges and extract useful information.
- **iropt_level** (*int*) – The optimization level of VEX IR (0, 1, 2). The default level will be used if *iropt_level* is None.
- **max_steps** (*int*) – The maximum number of basic blocks to recover forthe longest path from each start before pausing the recovery procedure.
- **state_add_options** – State options that will be added to the initial state.
- **state_remove_options** – State options that will be removed from the initial state.

**copy**()

Make a copy of the CFG.

**Return type**
*CFGEmulated*

**Returns**
A copy of the CFG instance.

**resume**(*starts=None*, *max_steps=None*)

Resume a paused or terminated control flow graph recovery.

**Parameters**

- **starts** (*iterable*) – A collection of new starts to resume from. If *starts* is None, we will resume CFG recovery from where it was paused before.
- **max_steps** (*int*) – The maximum number of blocks on the longest path starting from each start before pausing the recovery.

**Returns**
None

**remove_cycles**()

Forces graph to become acyclic, removes all loop back edges and edges between overlapped loop headers and their successors.

**downsize**()

Remove saved states from all CFGNodes to reduce memory usage.

**Returns**
None

**unroll_loops**(*max_loop_unrolling_times*)

Unroll loops for each function. The resulting CFG may still contain loops due to recursion, function calls, etc.

**Parameters**
**max_loop_unrolling_times** (*int*) – The maximum iterations of unrolling.

**Returns**
None

**force_unroll_loops**(*max_loop_unrolling_times*)

Unroll loops globally. The resulting CFG does not contain any loop, but this method is slow on large graphs.

**Parameters**
**max_loop_unrolling_times** (*int*) – The maximum iterations of unrolling.

**Returns**
None

**immediate_dominators**(*start*, *target_graph=None*)

Get all immediate dominators of sub graph from given node upwards.

**Parameters**

- **start** (*str*) – id of the node to navigate forwards from.
- **target_graph** (*networkx.classes.digraph.DiGraph*) – graph to analyse, default is self.graph.

**Returns**
each node of graph as index values, with element as respective node's immediate dominator.

**Return type**
dict

**immediate_postdominators**(*end*, *target_graph=None*)

Get all immediate postdominators of sub graph from given node upwards.

**Parameters**

- **start** (*str*) – id of the node to navigate forwards from.
- **target_graph** (*networkx.classes.digraph.DiGraph*) – graph to analyse, default is self.graph.

**Returns**
each node of graph as index values, with element as respective node's immediate dominator.

**Return type**
dict

**remove_fakerets**()

Get rid of fake returns (i.e., Ijk_FakeRet edges) from this CFG

**Returns**
None

**get_topological_order**(*cfg_node*)

Get the topological order of a CFG Node.

**Parameters**
**cfg_node** – A CFGNode instance.

**Returns**
An integer representing its order, or None if the CFGNode does not exist in the graph.

**get_subgraph**(*starting_node*, *block_addresses*)

Get a sub-graph out of a bunch of basic block addresses.

**Parameters**

- **starting_node** (CFGNode) – The beginning of the subgraph
- **block_addresses** (*iterable*) – A collection of block addresses that should be included in the subgraph if there is a path between *starting_node* and a CFGNode with the specified address, and all nodes on the path should also be included in the subgraph.

**Returns**
A new CFG that only contain the specific subgraph.

**Return type**
*CFGEmulated*

**get_function_subgraph**(*start*, *max_call_depth=None*)

Get a sub-graph of a certain function.

**Parameters**

- **start** – The function start. Currently it should be an integer.
- **max_call_depth** – Call depth limit. None indicates no limit.

**Returns**
A CFG instance which is a sub-graph of self.graph

**property context_sensitivity_level**

**property graph:** *SpillingCFG*

**property unresolvables**

Get those SimRuns that have non-resolvable exits.

**Returns**
A set of SimRuns

**Return type**
set

**property deadends**

Get all CFGNodes that has an out-degree of 0

**Returns**
A list of CFGNode instances

**Return type**
list

**class** angr.analyses.cfg.**CFGFast**

Bases: *ForwardAnalysis*[*CFGNode*, *CFGNode*, *CFGJob*, int, object], *CFGBase*

We find functions inside the given binary, and build a control-flow graph in very fast manners: instead of simulating program executions, keeping track of states, and performing expensive data-flow analysis, CFGFast will only perform light-weight analyses combined with some heuristics, and with some strong assumptions.

In order to identify as many functions as possible, and as accurate as possible, the following operation sequence is followed:

# Active scanning

- If the binary has "function symbols" (TODO: this term is not accurate enough), they are starting points of the code scanning
- If the binary does not have any "function symbol", we will first perform a function prologue scanning on the entire binary, and start from those places that look like function beginnings
- Otherwise, the binary's entry point will be the starting point for scanning

# Passive scanning

- After all active scans are done, we will go through the whole image and scan all code pieces

Due to the nature of those techniques that are used here, a base address is often not required to use this analysis routine. However, with a correct base address, CFG recovery will almost always yield a much better result. A custom analysis, called GirlScout, is specifically made to recover the base address of a binary blob. After the base address is determined, you may want to reload the binary with the new base address by creating a new Project object, and then re-recover the CFG.

```
PRINTABLES = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\
'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r'

SPECIAL_THUNKS = {'AMD64':
{b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x89\x04$\xc3':  ('jmp', 'rax'),
b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x8dd$\x08\xc3':  ('ret',)}}

tag:  str = 'CFGFast'

addr_type:  Literal['int', 'block_id', 'soot'] = 'int'
```

__init__(*binary=None*, *objects=None*, *regions=None*, *pickle_intermediate_results=False*, *symbols=True*, *function_prologues=None*, *resolve_indirect_jumps=True*, *force_segment=False*, *force_smart_scan=None*, *force_complete_scan=False*, *indirect_jump_target_limit=100000*, *data_references=True*, *cross_references=False*, *normalize=False*, *start_at_entry=True*, *function_starts=None*, *extra_memory_regions=None*, *data_type_guessing_handlers=None*, *arch_options=None*, *indirect_jump_resolvers=None*, *base_state=None*, *exclude_sparse_regions=True*, *skip_specific_regions=True*, *heuristic_plt_resolving=None*, *detect_tail_calls=False*, *low_priority=False*, *cfb=None*, *model=None*, *resume_state=None*, *eh_frame=True*, *eh_frame_boundaries=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=2048*, *nodecode_threshold=0.6*, *nodecode_step=16483*, *repeating_byte_run_threshold=64*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=True*, *treat_functions_as_complete=True*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=None*, *elf_eh_frame=None*, ***extra_arch_options*)

**Parameters**

- **binary** – The binary to recover CFG on. By default the main binary is used.
- **objects** – A list of objects to recover the CFG on. By default it will recover the CFG of all loaded objects.
- **regions** (*iterable*) – A list of tuples in the form of (start address, end address) describing memory regions that the CFG should cover.
- **pickle_intermediate_results** (*bool*) – If we want to store the intermediate results or not.
- **symbols** (*bool*) – Get function beginnings from symbols in the binary.
- **function_prologues** (bool | None) – Scan the binary for function prologues, and use those positions as function beginnings
- **resolve_indirect_jumps** (*bool*) – Try to resolve indirect jumps. This is necessary to resolve jump targets from jump tables, etc.
- **force_segment** (*bool*) – Force CFGFast to rely on binary segments instead of sections.
- **force_complete_scan** (*bool*) – Perform a complete scan on the binary and maximize the number of identified code blocks.
- **data_references** (*bool*) – Enables the collection of references to data used by individual instructions. This does not collect 'cross-references', particularly those that involve multiple instructions. For that, see *cross_references*
- **cross_references** (*bool*) – Whether CFGFast should collect "cross-references" from the entire program or not. This will populate the knowledge base with references to and from each recognizable address constant found in the code. Note that, because this performs constant propagation on the entire program, it may be much slower and consume more memory. This option implies *data_references=True*.
- **normalize** (*bool*) – Normalize the CFG as well as all function graphs after CFG recovery.
- **start_at_entry** (*bool*) – Begin CFG recovery at the entry point of this project. Setting it to False prevents CFGFast from viewing the entry point as one of the starting points of code scanning.
- **function_starts** (*list*) – A list of extra function starting points. CFGFast will try to resume scanning from each address in the list.

- **extra_memory_regions** (*list*) – A list of 2-tuple (start-address, end-address) that shows extra memory regions. Integers falling inside will be considered as pointers.
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **base_state** – A state to use as a backer for all memory loads
- **detect_tail_calls** (*bool*) – Enable aggressive tail-call optimization detection.
- **eh_frame** (*bool*) – Retrieve function starts (and maybe sizes later) from the .eh_frame of ELF binaries or exception records of PE binaries.
- **eh_frame_boundaries** (*bool*) – Treat function starts recorded in the .eh_frame of ELF binaries as hard function boundaries while tracing, the same way symbol addresses are treated. Without it, a tail-called function that is reached through a jump before it is reached through a call is absorbed into the caller. Only used if eh_frame is enabled.
- **skip_unmapped_addrs** – Ignore all branches into unmapped regions. True by default. You may want to set it to False if you are analyzing manually patched binaries or malware samples.
- **indirect_calls_always_return** (bool | None) – Should CFG assume indirect calls must return or not. Assuming indirect calls must return will significantly reduce the number of constant propagation runs, but may reduce the overall CFG recovery precision when facing non-returning indirect calls. By default, we only assume indirect calls always return for large binaries (region > 50KB).
- **jumptable_resolver_resolves_calls** (bool | None) – Whether JumpTableResolver should resolve indirect calls or not. Most indirect calls in C++ binaries or UEFI binaries cannot be resolved using jump table resolver and must be resolved using their specific resolvers. By default, we will only disable JumpTableResolver from resolving indirect calls for large binaries (region > 50 KB).
- **repeating_byte_run_threshold** – The minimum length of a run of one repeated byte that CFGFast refuses to decode as code. Such runs are filler, but many of them decode cleanly (e.g., 0x91 is *xchg ecx, eax* on x86), so linear disassembly would otherwise walk through them forever. Runs of the architecture's nop byte are exempt because execution really does flow through them. Set it to 0 to disable the check.
- **check_funcret_max_job** – When popping return-site jobs out of the job queue, angr will prioritize jobs for which the callee is known to return. This check may be slow when there are a large amount of jobs in different caller functions, and this situation often occurs in obfuscated binaries where many functions never return. This parameter acts as a threshold to disable this check when the number of jobs in the queue exceeds this threshold.
- **start** (*int*) – (Deprecated) The beginning address of CFG recovery.
- **end** (*int*) – (Deprecated) The end address of CFG recovery.
- **arch_options** (CFGArchOptions) – Architecture-specific options.
- **extra_arch_options** – Any key-value pair in kwargs will be seen as an arch-specific option and will be used to set the option value in self._arch_options.
- **retedges** (bool) – Whether to add return edges (from function endpoints to their return sites) in the CFG. Return edges are not added by default because they are often not useful during analysis; You can set retedges to True or call make_return_edges() after CFG recovery to create return edges. Note that this option does not impact function graphs.

- **model** – An existing CFGModel to write into. When a partial model from a previously aborted CFG recovery is passed in, recovery resumes on top of it: bytes covered by existing nodes and memory data are treated as already scanned and will not be re-lifted.
- **treat_functions_as_complete** (bool) – Whether post-analysis may treat the recovered functions as complete (drop bad functions, conclude returning=False for functions without return paths). Set to False for incremental runs that extend a partial model without completing it (see the resume recipes below).
- **resume_state** (*CFGResumeState* | None) – The resume_state captured by a previously aborted CFGFast instance. Its unprocessed jobs and unresolved indirect jumps are recreated (with their original function context) so that the recovery continues where it left off. Pass it together with model=.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress. The callback receives (percentage, text=..., cfg=...) where cfg is this CFGFast instance; calling cfg.abort() from the callback gracefully aborts the analysis: the core recovery loop stops at the next job boundary, post-analysis still runs and finalizes the partially recovered model, and cfg.should_abort remains True so callers can tell the model is partial.
- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.
- **force_smart_scan** (*bool | None*)
- **drop_bad_funcs** (*bool*)

Aborting and resuming CFG recovery:

After a graceful abort (see progress_callback above), self.resume_state captures the jobs that were still unprocessed (with their function context) and the unresolved indirect jumps; self.unprocessed_job_addrs holds the corresponding addresses for informational purposes. To resume recovery, create a new CFGFast instance reusing the partial model. To resume global scanning and finish the entire binary, re-inject the captured state while keeping the seeding options (symbols, function_prologues, eh_frame, force_smart_scan) at their original values:

```
proj.analyses.CFGFast(model=partial_cfg.model, start_at_entry=False,
                      resume_state=partial_cfg.resume_state)
```

Do not pass unprocessed_job_addrs as function_starts instead: function_starts seeds are treated as function heads, which would promote unprocessed call return sites and intra-function jump targets to functions.

To only resume from specific addresses (they must not be part of the partial model already; seeding an already-scanned address is a no-op):

```
proj.analyses.CFGFast(model=partial_cfg.model, function_starts=[addr], start_at_
↪entry=False,
                      symbols=False, function_prologues=False, eh_frame=False,
                      force_smart_scan=False, force_complete_scan=False,
                      treat_functions_as_complete=False)
```

treat_functions_as_complete=False tells post-analysis that the resulting model is still partial: functions must not be dropped as bad, and returning=False must not be concluded for (possibly truncated) functions - such conclusions would poison a later full resume the same way an abort would without them being reset.

**Returns**
None

**Parameters**

- **function_prologues** (*bool | None*)
- **force_smart_scan** (*bool | None*)
- **resume_state** (CFGResumeState | *None*)
- **indirect_calls_always_return** (*bool | None*)
- **jumptable_resolver_resolves_calls** (*bool | None*)
- **retedges** (*bool*)
- **drop_bad_funcs** (*bool*)
- **treat_functions_as_complete** (*bool*)

**unprocessed_job_addrs:** set[int]

**resume_state:** *CFGResumeState* | None

**stage:** str

**property graph:** *SpillingCFG*

**property memory_data**

**property jump_tables**

**property insn_addr_to_memory_data**

**do_full_xrefs**(*overlay_state=None*)

Perform xref recovery on all functions.

**Parameters**
**overlay** (SimState) – An overlay state for loading constant data.

**Returns**
None

**drop_bad_functions**()

**make_return_edges**()

For each returning function, create return edges in self.graph.

**Returns**
None

**copy**()

**output**()

**class** angr.analyses.cfg.**CFGFastSoot**

Bases: *CFGFast*

**addr_type:** Literal['int', 'block_id', 'soot'] = 'soot'

**drop_bad_functions**()

**make_functions**()

Revisit the entire control flow graph, create Function instances accordingly, and correctly put blocks into each function.

Although Function objects are crated during the CFG recovery, they are neither sound nor accurate. With a pre-constructed CFG, this method rebuilds all functions bearing the following rules:

- A block may only belong to one function.
- Small functions lying inside the startpoint and the endpoint of another function will be merged with the other function
- Tail call optimizations are detected.
- PLT stubs are aligned by 16.

**Returns**
None

## Submodules

| | |
|---|---|
| *cfb* | |
| *cfg* | |
| *cfg_arch_options* | |
| *cfg_base* | |
| *cfg_emulated* | |
| *cfg_fast* | |
| *cfg_fast_soot* | |
| *cfg_job_base* | |
| *go_prologue* | Recognition of the Go (gc) goroutine stack-growth preamble. |
| *indirect_jump_resolvers* | |
| *meta_structs* | |
| *pe_msvc_eh_structs* | Struct definitions and parsing utilities for MSVC C++ exception handling structures found in x86 PE binaries. |

### angr.analyses.cfg.cfb

**class** angr.analyses.cfg.cfb.**CFBlanketView**

Bases: object

A view into the control-flow blanket.

**__init__**(*cfb*)

**class** angr.analyses.cfg.cfb.**MemoryRegion**

Bases: object

**__init__**(*addr*, *size*, *type_*, *object_*, *cle_region*)

**class** angr.analyses.cfg.cfb.**Unknown**

Bases: object

An unknown byte region in a control-flow blanket.

**MAX_BYTES = 1616**

**__init__**(*addr*, *size*, *bytes_=None*, *object_=None*, *segment=None*, *section=None*, *loader=None*)

**property bytes**

The bytes of this region, for display purposes. Lazily loaded on first access (and capped at MAX_BYTES) when a loader is available.

**trimmed**(*new_start*, *new_size*)

Create a copy of this region covering [new_start, new_start + new_size). Bytes that were already loaded are sliced instead of being loaded again.

**Return type**
*Unknown*

**Parameters**

- **new_start** (*int*)
- **new_size** (*int*)

**class** angr.analyses.cfg.cfb.**CFBlanket**

Bases: *Analysis*

A Control-Flow Blanket is a representation for storing all instructions, data entries, and bytes of a full program.

Region types: - section - segment - extern - tls - kernel

**__init__**(*exclude_region_types=None*, *on_object_added=None*, *on_object_removed=None*)

**Parameters**

- **on_object_added** (Callable[[int, Any], None] | None) – Callable with parameters (addr, obj) called after an object is added to the blanket.
- **on_object_removed** (Callable[[int, Any], None] | None) – Callable with parameters (addr, obj) called after an object is removed from the blanket.
- **exclude_region_types** (*set[str] | None*)

**property regions**

Return all memory regions.

**floor_addr**(*addr*)

**Return type**
int

**Parameters**
**addr** (*int*)

**floor_item**(*addr*)

**Return type**
tuple[int, Any]

**Parameters**
**addr** (*int*)

**floor_items**(*addr=None*, *reverse=False*)

**ceiling_addr**(*addr*)

**Return type**
int

**Parameters**
**addr** (*int*)

`ceiling_item`(*addr*)

**Return type**
tuple[int, Any]

**Parameters**
**addr** (*int*)

`ceiling_items`(*addr=None*, *reverse=False*, *include_first=True*)

`add_obj`(*addr*, *obj*)

Add an object *obj* to the blanket at the specified address *addr*, keeping the blanket non-overlapping: existing objects that overlap [addr, addr + obj.size) are removed or trimmed, and the trimmed-off remainders are re-inserted. Objects without an integer size are stored as zero-span entries and never cause trimming.

`remove_obj`(*addr*, *fill=True*)

Remove the object at *addr* from the blanket. When *fill* is set (the default), the removed object's span is re-filled with an Unknown region so that the blanket remains a total cover of the mapped address space. Removing a missing address is a no-op.

**Returns**
The removed object, or None if no object exists at *addr*.

**Parameters**
**fill** (*bool*)

`add_function`(*func*)

Add a function *func* and all blocks of this function to the blanket.

`dbg_repr`()

The debugging representation of this CFBlanket.

**Returns**
The debugging representation of this CFBlanket.

**Return type**
str

### angr.analyses.cfg.cfg

**exception** angr.analyses.cfg.cfg.**OutdatedError**

Bases: Exception

**class** angr.analyses.cfg.cfg.**CFG**

Bases: *CFGFast*

tl;dr: CFG is just a wrapper around CFGFast for compatibility issues. It will be fully replaced by CFGFast in future releases. Feel free to use CFG if you intend to use CFGFast. Please use CFGEmulated if you *have to* use the old, slow, dynamically-generated version of CFG.

For multiple historical reasons, angr's CFG is accurate but slow, which does not meet what most people expect. We developed CFGFast for light-speed CFG recovery, and renamed the old CFG class to CFGEmulated. For compatibility concerns, CFG was kept as an alias to CFGEmulated.

However, so many new users of angr would load up a binary and generate a CFG immediately after running "pip install angr", and draw the conclusion that "angr's CFG is so slow - angr must be unusable!" Therefore, we made the hard decision: CFG will be an alias to CFGFast, instead of CFGEmulated.

To ease the transition of your existing code and script, the following changes are made:

- A CFG class, which is a sub class of CFGFast, is created.

- You will see both a warning message printed out to stderr and an exception raised by angr if you are passing CFG any parameter that only CFGEmulated supports. This exception is not a sub class of AngrError, so you wouldn't capture it with your old code by mistake.
- In the near future, this wrapper class will be removed completely, and CFG will be a simple alias to CFGFast.

We expect most interfaces are the same between CFGFast and CFGEmulated. Apparently some functionalities (like context-sensitivity, and state keeping) only exist in CFGEmulated, which is when you want to use CFGEmulated instead.

### angr.analyses.cfg.cfg_arch_options

**class** angr.analyses.cfg.cfg_arch_options.**CFGArchOptions**

Bases: object

Stores architecture-specific options and settings, as well as the detailed explanation of those options and settings.

Suppose *ao* is the CFGArchOptions object, and there is an option called *ret_jumpkind_heuristics*, you can access it by *ao.ret_jumpkind_heuristics* and set its value via *ao.ret_jumpkind_heuristics = True*

**Variables**

- **OPTIONS** (*dict*) – A dict of all default options for different architectures.
- **arch** (*archinfo.Arch*) – The architecture object.
- **_options** (*dict*) – Values of all CFG options that are specific to the current architecture.

**OPTIONS = {'ARMCortexM': {'has_arm_code': (<class 'bool'>, False), 'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, False)}, 'ARMEL': {'has_arm_code': (<class 'bool'>, True), 'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, True)}, 'ARMHF': {'has_arm_code': (<class 'bool'>, True), 'pattern_match_ifuncs': (<class 'bool'>, True), 'ret_jumpkind_heuristics': (<class 'bool'>, True), 'switch_mode_on_nodecode': (<class 'bool'>, True)}}**

**__init__**(*arch*, ***options*)

Constructor.

**Parameters**

- **arch** (*archinfo.Arch*) – The architecture instance.
- **options** (*dict*) – Architecture-specific options, which will be used to initialize this object.

**arch = None**

### angr.analyses.cfg.cfg_base

**class** angr.analyses.cfg.cfg_base.**CFGBase**

Bases: *Analysis*

The base class for control flow graphs.

**tag: str = None**

**addr_type: Literal['int', 'block_id', 'soot'] = None**

__init__(*sort*, *context_sensitivity_level*, *normalize=False*, *binary=None*, *objects=None*, *regions=None*, *exclude_sparse_regions=True*, *skip_specific_regions=True*, *force_segment=False*, *base_state=None*, *resolve_indirect_jumps=True*, *indirect_jump_resolvers=None*, *indirect_jump_target_limit=100000*, *detect_tail_calls=False*, *low_priority=False*, *skip_unmapped_addrs=True*, *sp_tracking_track_memory=True*, *model=None*)

**Parameters**

- **sort** (*str*) – 'fast' or 'emulated'.
- **context_sensitivity_level** (*int*) – The level of context-sensitivity of this CFG (see documentation for further details). It ranges from 0 to infinity.
- **normalize** (*bool*) – Whether the CFG as well as all Function graphs should be normalized.
- **binary** (*cle.backends.Backend*) – The binary to recover CFG on. By default, the main binary is used.
- **objects** – A list of objects to recover the CFG on. By default, it will recover the CFG of all loaded objects.
- **regions** (*iterable*) – A list of tuples in the form of (start address, end address) describing memory regions that the CFG should cover.
- **force_segment** (*bool*) – Force CFGFast to rely on binary segments instead of sections.
- **base_state** (*angr.SimState*) – A state to use as a backer for all memory loads.
- **resolve_indirect_jumps** (*bool*) – Whether to try to resolve indirect jumps. This is necessary to resolve jump targets from jump tables, etc.
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **indirect_jump_target_limit** (*int*) – Maximum indirect jump targets to be recovered.
- **skip_unmapped_addrs** – Ignore all branches into unmapped regions. True by default. You may want to set it to False if you are analyzing manually patched binaries or malware samples.
- **detect_tail_calls** (*bool*) – Aggressive tail-call optimization detection. This option is only respected in make_functions().
- **sp_tracking_track_memory** (*bool*) – Whether or not to track memory writes if tracking the stack pointer. This increases the accuracy of stack pointer tracking, especially for architectures without a base pointer. Only used if detect_tail_calls is enabled.
- **model** (*None or CFGModel*) – The CFGModel instance to write to. A new CFGModel instance will be created and registered with the knowledge base if *model* is None.

**Returns**

None

**indirect_jumps:** dict[int, *IndirectJump*]

**property model:** *CFGModel*

Get the CFGModel instance. :return: The CFGModel instance that this analysis currently uses.

**property normalized**

**property context_sensitivity_level**

**property functions**

A reference to the FunctionManager in the current knowledge base.

**Returns**
FunctionManager with all functions

**Return type**
angr.knowledge_plugins.FunctionManager

**property regions: list[tuple[int, int]]**

The memory regions that this analysis covers. An empty list means it had nothing to scan.

**Returns**
A sorted list of (start address, end address) tuples.

**make_copy**(*copy_to*)

Copy self attributes to the new object.

**Parameters**
**copy_to** (CFGBase) – The target to copy to.

**Returns**
None

**copy**()

**output**()

**generate_index**()

Generate an index of all nodes in the graph in order to speed up get_any_node() with anyaddr=True.

**Returns**
None

**get_loop_back_edges**()

**property graph: *SpillingCFG***

**remove_edge**(*block_from*, *block_to*)

**is_thumb_addr**(*addr*)

**record_memory_data_addr**(*addr*)

Record the address of a newly added memory data object.

**Return type**
None

**Parameters**
**addr** (*int*)

**reset_memory_data_addrs**()

Reset the set of addresses of newly added memory data objects.

**Return type**
None

**normalize**()

Normalize the CFG, making sure that there are no overlapping basic blocks.

Note that this method will not alter transition graphs of each function in self.kb.functions. You may call normalize() on each Function object to normalize their transition graphs.

**Returns**
None

**mark_function_alignments**()

Find all potential function alignments and mark them.

Note that it is not always correct to simply remove them, because these functions may not be actual alignments but part of an actual function, and is incorrectly marked as an individual function because of failures in resolving indirect jumps. An example is in the test binary `x86_64/dir_gcc_-O0` 0x40541d (indirect jump at 0x4051b0). If the indirect jump cannot be correctly resolved, removing function 0x40541d will cause a missing label failure in reassembler.

**Returns**
None

**make_functions**()

Revisit the entire control flow graph, create Function instances accordingly, and correctly put blocks into each function.

Although Function objects are created during the CFG recovery, they are neither sound nor accurate. With a pre-constructed CFG, this method rebuilds all functions bearing the following rules:

- A block may only belong to one function.
- Small functions lying inside the startpoint and the endpoint of another function will be merged with the other function
- Tail call optimizations are detected.
- PLT stubs are aligned by 16.

**Returns**
None

### angr.analyses.cfg.cfg_emulated

**class** angr.analyses.cfg.cfg_emulated.**CFGJob**

Bases: *CFGJobBase*

The job class that CFGEmulated uses.

**__init__**(**args*, ***kwargs*)

**property block_id**

**property is_syscall**

**class** angr.analyses.cfg.cfg_emulated.**PendingJob**

Bases: object

A PendingJob is whatever will be put into our pending_exit list. A pending exit is an entry that created by the returning of a call or syscall. It is "pending" since we cannot immediately figure out whether this entry will be executed or not. If the corresponding call/syscall intentionally doesn't return, then the pending exit will be removed. If the corresponding call/syscall returns, then the pending exit will be removed as well (since a real entry is created from the returning and will be analyzed later). If the corresponding call/syscall might return, but

for some reason (for example, an unsupported instruction is met during the analysis) our analysis does not return properly, then the pending exit will be picked up and put into remaining_jobs list.

**__init__**(*caller_func_addr*, *returning_source*, *state*, *src_block_id*, *src_exit_stmt_idx*, *src_exit_ins_addr*, *call_stack*)

**Parameters**

- **returning_source** – Address of the callee function. It might be None if address of the callee is not resolvable.
- **state** – The state after returning from the callee function. Of course there is no way to get a precise state without emulating the execution of the callee, but at least we can properly adjust the stack and registers to imitate the real returned state.
- **call_stack** – A callstack.

**class** angr.analyses.cfg.cfg_emulated.**CFGEmulated**

Bases: *ForwardAnalysis*, *CFGBase*

This class represents a control-flow graph.

**tag:** `str = 'CFGEmulated'`

**addr_type:** `Literal['int', 'block_id', 'soot'] = 'block_id'`

**__init__**(*context_sensitivity_level=1*, *start=None*, *avoid_runs=None*, *enable_function_hints=False*, *call_depth=None*, *call_tracing_filter=None*, *initial_state=None*, *starts=None*, *keep_state=False*, *indirect_jump_target_limit=100000*, *resolve_indirect_jumps=True*, *enable_advanced_backward_slicing=False*, *enable_symbolic_back_traversal=False*, *indirect_jump_resolvers=None*, *additional_edges=None*, *no_construct=False*, *normalize=False*, *max_iterations=1*, *address_whitelist=None*, *base_graph=None*, *iropt_level=None*, *max_steps=None*, *state_add_options=None*, *state_remove_options=None*, *model=None*)

All parameters are optional.

**Parameters**

- **context_sensitivity_level** – The level of context-sensitivity of this CFG (see documentation for further details). It ranges from 0 to infinity. Default 1.
- **avoid_runs** – A list of runs to avoid.
- **enable_function_hints** – Whether to use function hints (constants that might be used as exit targets) or not.
- **call_depth** – How deep in the call stack to trace.
- **call_tracing_filter** – Filter to apply on a given path and jumpkind to determine if it should be skipped when call_depth is reached.
- **initial_state** – An initial state to use to begin analysis.
- **starts** (*iterable*) – A collection of starting points to begin analysis. It can contain the following three different types of entries: an address specified as an integer, a 2-tuple that includes an integer address and a jumpkind, or a SimState instance. Unsupported entries in starts will lead to an AngrCFGError being raised.
- **keep_state** – Whether to keep the SimStates for each CFGNode.
- **resolve_indirect_jumps** – Whether to enable the indirect jump resolvers for resolving indirect jumps

- **enable_advanced_backward_slicing** – Whether to enable an intensive technique for resolving indirect jumps
- **enable_symbolic_back_traversal** – Whether to enable an intensive technique for resolving indirect jumps
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **additional_edges** – A dict mapping addresses of basic blocks to addresses of successors to manually include and analyze forward from.
- **no_construct** (*bool*) – Skip the construction procedure. Only used in unit-testing.
- **normalize** (*bool*) – If the CFG as well as all Function graphs should be normalized or not.
- **max_iterations** (*int*) – The maximum number of iterations that each basic block should be "executed". 1 by default. Larger numbers of iterations are usually required for complex analyses like loop analysis.
- **address_whitelist** (*iterable*) – A list of allowed addresses. Any basic blocks outside of this collection of addresses will be ignored.
- **base_graph** (*networkx.DiGraph*) – A basic control flow graph to follow. Each node inside this graph must have the following properties: *addr* and *size*. CFG recovery will strictly follow nodes and edges shown in the graph, and discard any control flow that does not follow an existing edge in the base graph. For example, you can pass in a Function local transition graph as the base graph, and CFGEmulated will traverse nodes and edges and extract useful information.
- **iropt_level** (*int*) – The optimization level of VEX IR (0, 1, 2). The default level will be used if *iropt_level* is None.
- **max_steps** (*int*) – The maximum number of basic blocks to recover forthe longest path from each start before pausing the recovery procedure.
- **state_add_options** – State options that will be added to the initial state.
- **state_remove_options** – State options that will be removed from the initial state.

**copy**()

Make a copy of the CFG.

**Return type**
*CFGEmulated*

**Returns**
A copy of the CFG instance.

**resume**(*starts=None*, *max_steps=None*)

Resume a paused or terminated control flow graph recovery.

**Parameters**

- **starts** (*iterable*) – A collection of new starts to resume from. If *starts* is None, we will resume CFG recovery from where it was paused before.
- **max_steps** (*int*) – The maximum number of blocks on the longest path starting from each start before pausing the recovery.

**Returns**
None

`remove_cycles`()

Forces graph to become acyclic, removes all loop back edges and edges between overlapped loop headers and their successors.

`downsize`()

Remove saved states from all CFGNodes to reduce memory usage.

**Returns**
None

`unroll_loops`(*max_loop_unrolling_times*)

Unroll loops for each function. The resulting CFG may still contain loops due to recursion, function calls, etc.

**Parameters**
**max_loop_unrolling_times** (*int*) – The maximum iterations of unrolling.

**Returns**
None

`force_unroll_loops`(*max_loop_unrolling_times*)

Unroll loops globally. The resulting CFG does not contain any loop, but this method is slow on large graphs.

**Parameters**
**max_loop_unrolling_times** (*int*) – The maximum iterations of unrolling.

**Returns**
None

`immediate_dominators`(*start*, *target_graph=None*)

Get all immediate dominators of sub graph from given node upwards.

**Parameters**

- **start** (*str*) – id of the node to navigate forwards from.
- **target_graph** (*networkx.classes.digraph.DiGraph*) – graph to analyse, default is self.graph.

**Returns**
each node of graph as index values, with element as respective node's immediate dominator.

**Return type**
dict

`immediate_postdominators`(*end*, *target_graph=None*)

Get all immediate postdominators of sub graph from given node upwards.

**Parameters**

- **start** (*str*) – id of the node to navigate forwards from.
- **target_graph** (*networkx.classes.digraph.DiGraph*) – graph to analyse, default is self.graph.

**Returns**
each node of graph as index values, with element as respective node's immediate dominator.

**Return type**
dict

**remove_fakerets**()

Get rid of fake returns (i.e., Ijk_FakeRet edges) from this CFG

**Returns**
None

**get_topological_order**(*cfg_node*)

Get the topological order of a CFG Node.

**Parameters**
**cfg_node** – A CFGNode instance.

**Returns**
An integer representing its order, or None if the CFGNode does not exist in the graph.

**get_subgraph**(*starting_node*, *block_addresses*)

Get a sub-graph out of a bunch of basic block addresses.

**Parameters**

- **starting_node** (CFGNode) – The beginning of the subgraph
- **block_addresses** (*iterable*) – A collection of block addresses that should be included in the subgraph if there is a path between *starting_node* and a CFGNode with the specified address, and all nodes on the path should also be included in the subgraph.

**Returns**
A new CFG that only contain the specific subgraph.

**Return type**
*CFGEmulated*

**get_function_subgraph**(*start*, *max_call_depth=None*)

Get a sub-graph of a certain function.

**Parameters**

- **start** – The function start. Currently it should be an integer.
- **max_call_depth** – Call depth limit. None indicates no limit.

**Returns**
A CFG instance which is a sub-graph of self.graph

**property context_sensitivity_level**

**property graph:** *SpillingCFG*

**property unresolvables**

Get those SimRuns that have non-resolvable exits.

**Returns**
A set of SimRuns

**Return type**
set

**property deadends**

Get all CFGNodes that has an out-degree of 0

**Returns**
A list of CFGNode instances

**Return type**
list

### angr.analyses.cfg.cfg_fast

**exception** angr.analyses.cfg.cfg_fast.**ContinueScanningNotification**

Bases: RuntimeError

A notification raised by _next_code_addr_core() to indicate no code address is found and _next_code_addr_core() should be invoked again.

**class** angr.analyses.cfg.cfg_fast.**ARMDecodingMode**

Bases: object

Enums indicating decoding mode for ARM code.

**ARM = 0**

**THUMB = 1**

**class** angr.analyses.cfg.cfg_fast.**DecodingAssumption**

Bases: object

Describes the decoding mode (ARM/THUMB) for a given basic block identified by its address.

**__init__**(*addr*, *size*, *mode*)

**Parameters**

- **addr** (*int*)
- **size** (*int*)
- **mode** (*int*)

**add_data_seg**(*addr*, *size*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**class** angr.analyses.cfg.cfg_fast.**FunctionReturn**

Bases: object

FunctionReturn describes a function call in a specific location and its return location. Hashable and equatable

**__init__**(*callee_func_addr*, *caller_func_addr*, *call_site_addr*, *return_to*)

**callee_func_addr**

**caller_func_addr**

**call_site_addr**

**return_to**

class angr.analyses.cfg.cfg_fast.**PendingJobs**

Bases: object

A collection of pending jobs during CFG recovery.

**__init__**(*kb*, *deregister_job_callback*)

**add_job**(*job*)

**all_jobs**()

Iterate over all pending jobs.

**Returns**
An iterator over all pending CFGJob instances.

**pop_job**(*returning=True*)

Pop a job from the pending jobs list.

When returning == True, we prioritize the jobs whose functions are known to be returning (function.returning is True). As an optimization, we are sorting the pending jobs list according to job.function.returning.

**Parameters**
**returning** (*bool*) – Only pop a pending job if the corresponding function returns.

**Returns**
A pending job if we can find one, or None if we cannot find any that satisfies the requirement.

**Return type**
*angr.analyses.cfg.cfg_fast.CFGJob*

**cleanup**()

Remove those pending exits if: a) they are the return exits of non-returning SimProcedures b) they are the return exits of non-returning syscalls b) they are the return exits of non-returning functions

**Returns**
None

**add_returning_function**(*func_addr*)

Mark a function as returning.

**Parameters**
**func_addr** (*int*) – Address of the function that returns.

**Returns**
None

**add_nonreturning_function**(*func_addr*)

Mark a function as not returning.

**Parameters**
**func_addr** (*int*) – Address of the function that does not return.

**Returns**
None

**clear_updated_functions**()

Clear the updated_functions set.

**Returns**
None

class angr.analyses.cfg.cfg_fast.**ResumeJob**

Bases: NamedTuple

Describes an unprocessed CFGJob captured when a CFG recovery is aborted, with enough context to faithfully re-create the job when the recovery is resumed: func_addr preserves the function that the job belongs to (so that resuming does not promote mid-function block addresses, e.g. call return sites, to function heads), and the source block and function edges are stored as addresses (so that no CFGNode references survive across analyses) and are re-resolved against the model when the job is re-created.

**addr:** int

Alias for field number 0

**func_addr:** int

Alias for field number 1

**jumpkind:** str

Alias for field number 2

**returning_source:** int | None

Alias for field number 3

**syscall:** bool

Alias for field number 4

**gp:** int | None

Alias for field number 5

**last_addr:** int | None

Alias for field number 6

**src_node_addr:** int | None

Alias for field number 7

**src_ins_addr:** int | None

Alias for field number 8

**src_stmt_idx:** int | None

Alias for field number 9

**func_edges:** tuple[tuple, ...]

Alias for field number 10

static **__new__**(*_cls*, *addr: int*, *func_addr: int*, *jumpkind: str*, *returning_source: int | None = None*, *syscall: bool = False*, *gp: int | None = None*, *last_addr: int | None = None*, *src_node_addr: int | None = None*, *src_ins_addr: int | None = None*, *src_stmt_idx: int | None = None*, *func_edges: tuple[tuple, ...] = ()*)

Create new instance of ResumeJob(addr, func_addr, jumpkind, returning_source, syscall, gp, last_addr, src_node_addr, src_ins_addr, src_stmt_idx, func_edges)

**Parameters**

- **addr** (*int*)
- **func_addr** (*int*)
- **jumpkind** (*str*)
- **returning_source** (*int | None*)
- **syscall** (*bool*)

- **gp** (*int | None*)
- **last_addr** (*int | None*)
- **src_node_addr** (*int | None*)
- **src_ins_addr** (*int | None*)
- **src_stmt_idx** (*int | None*)
- **func_edges** (*tuple[tuple, ...]*)

**class** angr.analyses.cfg.cfg_fast.**CFGResumeState**

Bases: object

Captured state of an aborted CFG recovery. Pass it to a new CFGFast instance (via the resume_state parameter, together with model=) to resume the recovery where it left off.

**__init__**(*jobs=None*, *indirect_jumps=None*)

**Parameters**

- **jobs** (*list[ResumeJob] | None*)
- **indirect_jumps** (*list[IndirectJump] | None*)

**jobs:** list[*ResumeJob*]

**indirect_jumps:** list[*IndirectJump*]

**class** angr.analyses.cfg.cfg_fast.**FunctionEdge**

Bases: object

Describes an edge in functions' transition graphs. Base class for all types of edges.

**apply**(*cfg*)

**ins_addr**

**src_func_addr**

**stmt_idx**

**class** angr.analyses.cfg.cfg_fast.**FunctionTransitionEdge**

Bases: *FunctionEdge*

Describes a transition edge in functions' transition graphs.

**__init__**(*src_node*, *dst_addr*, *src_func_addr*, *to_outside=False*, *dst_func_addr=None*, *stmt_idx=None*, *ins_addr=None*, *is_exception=False*)

**src_node**

**dst_addr**

**to_outside**

**dst_func_addr**

**is_exception**

**apply**(*cfg*)

class angr.analyses.cfg.cfg_fast.**FunctionCallEdge**

Bases: *FunctionEdge*

Describes a call edge in functions' transition graphs.

**__init__**(*src_node*, *dst_addr*, *ret_addr*, *src_func_addr*, *syscall=False*, *stmt_idx=None*, *ins_addr=None*)

**src_node**

**dst_addr**

**ret_addr**

**syscall**

**apply**(*cfg*)

class angr.analyses.cfg.cfg_fast.**FunctionFakeRetEdge**

Bases: *FunctionEdge*

Describes a FakeReturn (also called fall-through) edge in functions' transition graphs.

**__init__**(*src_node*, *dst_addr*, *src_func_addr*, *confirmed=None*)

**src_node**

**dst_addr**

**confirmed**

**apply**(*cfg*)

class angr.analyses.cfg.cfg_fast.**FunctionReturnEdge**

Bases: *FunctionEdge*

Describes a return (from a function call or a syscall) edge in functions' transition graphs.

**__init__**(*ret_from_addr*, *ret_to_addr*, *dst_func_addr*)

**ret_from_addr**

**ret_to_addr**

**dst_func_addr**

**apply**(*cfg*)

class angr.analyses.cfg.cfg_fast.**CFGJobType**

Bases: Enum

Defines the type of work of a CFGJob

**NORMAL = 0**

**FUNCTION_PROLOGUE = 1**

**COMPLETE_SCANNING = 2**

**IFUNC_HINTS = 3**

**DATAREF_HINTS = 4**

**EH_FRAME_HINTS = 5**

**class** angr.analyses.cfg.cfg_fast.**CFGJob**

Bases: object

Defines a job to work on during the CFG recovery

**__init__**(*addr*, *func_addr*, *jumpkind*, *ret_target=None*, *last_addr=None*, *src_node=None*, *src_ins_addr=None*, *src_stmt_idx=None*, *returning_source=None*, *syscall=False*, *func_edges=None*, *job_type=CFGJobType.NORMAL*, *gp=None*)

**Parameters**

- **addr** (*int*)
- **func_addr** (*int*)
- **jumpkind** (*str*)
- **ret_target** (*int | None*)
- **last_addr** (*int | None*)
- **src_node** (CFGNode | *None*)
- **src_ins_addr** (*int | None*)
- **src_stmt_idx** (*int | None*)
- **syscall** (*bool*)
- **func_edges** (*list | None*)
- **job_type** (CFGJobType)
- **gp** (*int | None*)

**addr**

**func_addr**

**jumpkind**

**ret_target**

**last_addr**

**src_node**

**src_ins_addr**

**src_stmt_idx**

**returning_source**

**syscall**

**job_type**

**gp**

**add_function_edge**(*edge*)

**apply_function_edges**(*cfg*, *clear=False*)

class angr.analyses.cfg.cfg_fast.**CFGFast**

Bases: *ForwardAnalysis*[*CFGNode*, *CFGNode*, *CFGJob*, int, object], *CFGBase*

We find functions inside the given binary, and build a control-flow graph in very fast manners: instead of simulating program executions, keeping track of states, and performing expensive data-flow analysis, CFGFast will only perform light-weight analyses combined with some heuristics, and with some strong assumptions.

In order to identify as many functions as possible, and as accurate as possible, the following operation sequence is followed:

# Active scanning

- If the binary has "function symbols" (TODO: this term is not accurate enough), they are starting points of the code scanning
- If the binary does not have any "function symbol", we will first perform a function prologue scanning on the entire binary, and start from those places that look like function beginnings
- Otherwise, the binary's entry point will be the starting point for scanning

# Passive scanning

- After all active scans are done, we will go through the whole image and scan all code pieces

Due to the nature of those techniques that are used here, a base address is often not required to use this analysis routine. However, with a correct base address, CFG recovery will almost always yield a much better result. A custom analysis, called GirlScout, is specifically made to recover the base address of a binary blob. After the base address is determined, you may want to reload the binary with the new base address by creating a new Project object, and then re-recover the CFG.

**PRINTABLES = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&**
**'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r'**

**SPECIAL_THUNKS = {'AMD64':**
**{b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x89\x04$\xc3': ('jmp', 'rax'),**
**b'\xe8\x07\x00\x00\x00\xf3\x90\x0f\xae\xe8\xeb\xf9H\x8dd$\x08\xc3': ('ret',)}}**

**tag: str = 'CFGFast'**

**addr_type: Literal['int', 'block_id', 'soot'] = 'int'**

**__init__**(*binary=None*, *objects=None*, *regions=None*, *pickle_intermediate_results=False*, *symbols=True*, *function_prologues=None*, *resolve_indirect_jumps=True*, *force_segment=False*, *force_smart_scan=None*, *force_complete_scan=False*, *indirect_jump_target_limit=100000*, *data_references=True*, *cross_references=False*, *normalize=False*, *start_at_entry=True*, *function_starts=None*, *extra_memory_regions=None*, *data_type_guessing_handlers=None*, *arch_options=None*, *indirect_jump_resolvers=None*, *base_state=None*, *exclude_sparse_regions=True*, *skip_specific_regions=True*, *heuristic_plt_resolving=None*, *detect_tail_calls=False*, *low_priority=False*, *cfb=None*, *model=None*, *resume_state=None*, *eh_frame=True*, *eh_frame_boundaries=True*, *exceptions=True*, *skip_unmapped_addrs=True*, *nodecode_window_size=2048*, *nodecode_threshold=0.6*, *nodecode_step=16483*, *repeating_byte_run_threshold=64*, *check_funcret_max_job=500*, *indirect_calls_always_return=None*, *jumptable_resolver_resolves_calls=None*, *retedges=False*, *drop_bad_funcs=True*, *treat_functions_as_complete=True*, *start=None*, *end=None*, *collect_data_references=None*, *extra_cross_references=None*, *elf_eh_frame=None*, ***extra_arch_options*)

**Parameters**

- **binary** – The binary to recover CFG on. By default the main binary is used.

- **objects** – A list of objects to recover the CFG on. By default it will recover the CFG of all loaded objects.
- **regions** (*iterable*) – A list of tuples in the form of (start address, end address) describing memory regions that the CFG should cover.
- **pickle_intermediate_results** (*bool*) – If we want to store the intermediate results or not.
- **symbols** (*bool*) – Get function beginnings from symbols in the binary.
- **function_prologues** (bool | None) – Scan the binary for function prologues, and use those positions as function beginnings
- **resolve_indirect_jumps** (*bool*) – Try to resolve indirect jumps. This is necessary to resolve jump targets from jump tables, etc.
- **force_segment** (*bool*) – Force CFGFast to rely on binary segments instead of sections.
- **force_complete_scan** (*bool*) – Perform a complete scan on the binary and maximize the number of identified code blocks.
- **data_references** (*bool*) – Enables the collection of references to data used by individual instructions. This does not collect 'cross-references', particularly those that involve multiple instructions. For that, see *cross_references*
- **cross_references** (*bool*) – Whether CFGFast should collect "cross-references" from the entire program or not. This will populate the knowledge base with references to and from each recognizable address constant found in the code. Note that, because this performs constant propagation on the entire program, it may be much slower and consume more memory. This option implies *data_references=True*.
- **normalize** (*bool*) – Normalize the CFG as well as all function graphs after CFG recovery.
- **start_at_entry** (*bool*) – Begin CFG recovery at the entry point of this project. Setting it to False prevents CFGFast from viewing the entry point as one of the starting points of code scanning.
- **function_starts** (*list*) – A list of extra function starting points. CFGFast will try to resume scanning from each address in the list.
- **extra_memory_regions** (*list*) – A list of 2-tuple (start-address, end-address) that shows extra memory regions. Integers falling inside will be considered as pointers.
- **indirect_jump_resolvers** (*list*) – A custom list of indirect jump resolvers. If this list is None or empty, default indirect jump resolvers specific to this architecture and binary types will be loaded.
- **base_state** – A state to use as a backer for all memory loads
- **detect_tail_calls** (*bool*) – Enable aggressive tail-call optimization detection.
- **eh_frame** (*bool*) – Retrieve function starts (and maybe sizes later) from the .eh_frame of ELF binaries or exception records of PE binaries.
- **eh_frame_boundaries** (*bool*) – Treat function starts recorded in the .eh_frame of ELF binaries as hard function boundaries while tracing, the same way symbol addresses are treated. Without it, a tail-called function that is reached through a jump before it is reached through a call is absorbed into the caller. Only used if eh_frame is enabled.
- **skip_unmapped_addrs** – Ignore all branches into unmapped regions. True by default. You may want to set it to False if you are analyzing manually patched binaries or malware samples.

- **indirect_calls_always_return** (bool | None) – Should CFG assume indirect calls must return or not. Assuming indirect calls must return will significantly reduce the number of constant propagation runs, but may reduce the overall CFG recovery precision when facing non-returning indirect calls. By default, we only assume indirect calls always return for large binaries (region > 50KB).
- **jumptable_resolver_resolves_calls** (bool | None) – Whether JumpTableResolver should resolve indirect calls or not. Most indirect calls in C++ binaries or UEFI binaries cannot be resolved using jump table resolver and must be resolved using their specific resolvers. By default, we will only disable JumpTableResolver from resolving indirect calls for large binaries (region > 50 KB).
- **repeating_byte_run_threshold** – The minimum length of a run of one repeated byte that CFGFast refuses to decode as code. Such runs are filler, but many of them decode cleanly (e.g., 0x91 is *xchg ecx, eax* on x86), so linear disassembly would otherwise walk through them forever. Runs of the architecture's nop byte are exempt because execution really does flow through them. Set it to 0 to disable the check.
- **check_funcret_max_job** – When popping return-site jobs out of the job queue, angr will prioritize jobs for which the callee is known to return. This check may be slow when there are a large amount of jobs in different caller functions, and this situation often occurs in obfuscated binaries where many functions never return. This parameter acts as a threshold to disable this check when the number of jobs in the queue exceeds this threshold.
- **start** (*int*) – (Deprecated) The beginning address of CFG recovery.
- **end** (*int*) – (Deprecated) The end address of CFG recovery.
- **arch_options** (CFGArchOptions) – Architecture-specific options.
- **extra_arch_options** – Any key-value pair in kwargs will be seen as an arch-specific option and will be used to set the option value in self._arch_options.
- **retedges** (bool) – Whether to add return edges (from function endpoints to their return sites) in the CFG. Return edges are not added by default because they are often not useful during analysis; You can set retedges to True or call make_return_edges() after CFG recovery to create return edges. Note that this option does not impact function graphs.
- **model** – An existing CFGModel to write into. When a partial model from a previously aborted CFG recovery is passed in, recovery resumes on top of it: bytes covered by existing nodes and memory data are treated as already scanned and will not be re-lifted.
- **treat_functions_as_complete** (bool) – Whether post-analysis may treat the recovered functions as complete (drop bad functions, conclude returning=False for functions without return paths). Set to False for incremental runs that extend a partial model without completing it (see the resume recipes below).
- **resume_state** (*CFGResumeState* | None) – The resume_state captured by a previously aborted CFGFast instance. Its unprocessed jobs and unresolved indirect jumps are recreated (with their original function context) so that the recovery continues where it left off. Pass it together with model=.
- **progress_callback** – (Inherited from angr.Analysis.) Callback for CFG recovery progress. The callback receives (percentage, text=..., cfg=...) where cfg is this CFGFast instance; calling cfg.abort() from the callback gracefully aborts the analysis: the core recovery loop stops at the next job boundary, post-analysis still runs and finalizes the partially recovered model, and cfg.should_abort remains True so callers can tell the model is partial.

- **show_progressbar** (*bool*) – (Inherited from angr.Analysis.) Show a progressbar during CFG recovery.
- **force_smart_scan** (*bool | None*)
- **drop_bad_funcs** (*bool*)

Aborting and resuming CFG recovery:

After a graceful abort (see progress_callback above), self.resume_state captures the jobs that were still unprocessed (with their function context) and the unresolved indirect jumps; self.unprocessed_job_addrs holds the corresponding addresses for informational purposes. To resume recovery, create a new CFGFast instance reusing the partial model. To resume global scanning and finish the entire binary, re-inject the captured state while keeping the seeding options (symbols, function_prologues, eh_frame, force_smart_scan) at their original values:

```
proj.analyses.CFGFast(model=partial_cfg.model, start_at_entry=False,
                      resume_state=partial_cfg.resume_state)
```

Do not pass unprocessed_job_addrs as function_starts instead: function_starts seeds are treated as function heads, which would promote unprocessed call return sites and intra-function jump targets to functions.

To only resume from specific addresses (they must not be part of the partial model already; seeding an already-scanned address is a no-op):

```
proj.analyses.CFGFast(model=partial_cfg.model, function_starts=[addr], start_at_
↪entry=False,
                      symbols=False, function_prologues=False, eh_frame=False,
                      force_smart_scan=False, force_complete_scan=False,
                      treat_functions_as_complete=False)
```

treat_functions_as_complete=False tells post-analysis that the resulting model is still partial: functions must not be dropped as bad, and returning=False must not be concluded for (possibly truncated) functions - such conclusions would poison a later full resume the same way an abort would without them being reset.

**Returns**
None

**Parameters**

- **function_prologues** (*bool | None*)
- **force_smart_scan** (*bool | None*)
- **resume_state** (*CFGResumeState | None*)
- **indirect_calls_always_return** (*bool | None*)
- **jumptable_resolver_resolves_calls** (*bool | None*)
- **retedges** (*bool*)
- **drop_bad_funcs** (*bool*)
- **treat_functions_as_complete** (*bool*)

**unprocessed_job_addrs:** `set[int]`

**resume_state:** *CFGResumeState* | `None`

**stage:** `str`

property graph: *SpillingCFG*

property memory_data

property jump_tables

property insn_addr_to_memory_data

do_full_xrefs(*overlay_state=None*)

Perform xref recovery on all functions.

**Parameters**
**overlay** (SimState) – An overlay state for loading constant data.

**Returns**
None

drop_bad_functions()

make_return_edges()

For each returning function, create return edges in self.graph.

**Returns**
None

copy()

output()

### angr.analyses.cfg.cfg_fast_soot

class angr.analyses.cfg.cfg_fast_soot.**CFGFastSoot**

Bases: *CFGFast*

addr_type: Literal['int', 'block_id', 'soot'] = 'soot'

drop_bad_functions()

make_functions()

Revisit the entire control flow graph, create Function instances accordingly, and correctly put blocks into each function.

Although Function objects are crated during the CFG recovery, they are neither sound nor accurate. With a pre-constructed CFG, this method rebuilds all functions bearing the following rules:

- A block may only belong to one function.
- Small functions lying inside the startpoint and the endpoint of another function will be merged with the other function
- Tail call optimizations are detected.
- PLT stubs are aligned by 16.

**Returns**
None

### angr.analyses.cfg.cfg_job_base

**class** angr.analyses.cfg.cfg_job_base.**FunctionKey**

Bases: object

A context-sensitive key for a function.

**__init__**(*addr*, *callsite_tuples*)

**callsite_repr**()

**static new**(*addr*, *callsite_tuples*)

**class** angr.analyses.cfg.cfg_job_base.**CFGJobBase**

Bases: object

Describes an entry in CFG or VFG. Only used internally by the analysis.

**__init__**(*addr*, *state*, *context_sensitivity_level*, *block_id=None*, *src_block_id=None*, *src_exit_stmt_idx=None*, *src_ins_addr=None*, *jumpkind=None*, *call_stack=None*, *is_narrowing=False*, *skip=False*, *final_return_address=None*)

**Parameters**

- **state** (SimState)
- **jumpkind** (*str | None*)

**property call_stack**

**call_stack_copy**()

**get_call_stack_suffix**()

**property func_addr**

**property current_stack_pointer**

### angr.analyses.cfg.go_prologue

Recognition of the Go (gc) goroutine stack-growth preamble.

Every non-nosplit Go function starts with a check of the current stack pointer (or of the prospective frame bottom) against `g.stackguard0`, followed by a forward branch to a `runtime.morestack` trampoline appended after the function body. The conventional prologue the architecture's `function_prologs` describe only comes *after* that check, so a plain prologue scan reports function starts several instructions past the real entry.

`g.stackguard0` sits at `2 * ptrSize` in `g` – `[reg + 0x10]` on 64-bit targets, `[reg + 8]` on 32-bit ones. Functions the compiler marks as C functions (`//go:systemstack` and friends) check `g.stackguard1` instead, one pointer further in. The register holding `g` is architectural on most targets (r14 on amd64, x28 on arm64, r10 on arm), so the compare alone identifies the check. 386 has no dedicated g register and reaches `g` through TLS with an OS-specific sequence, which is therefore part of the pattern there.

The encodings have drifted across releases, so the tables cover several forms per architecture: windows/386 reaches g through `fs:[0x14]` up to go1.20 and through `runtime.tls_g` from go1.21, and the arm64 scratch pair was x1/x2 in go1.17 and is x16/x17 from go1.18 on – where go1.18 still compared through x17 and go1.19 and later compare sp directly.

angr.analyses.cfg.go_prologue.**go_preamble_supported**(*arch_name*)

Whether the Go stack-growth preamble is recognized on the named architecture.

**Return type**
bool

**Parameters**
**arch_name** (*str*)

angr.analyses.cfg.go_prologue.**find_go_stack_check_preambles**(*blob*, *arch_name*, *base=0*)

Find every Go goroutine stack-growth preamble in `blob`.

**Parameters**

- **blob** (bytes) – A blob of machine code.
- **arch_name** (str) – `archinfo` name of the architecture `blob` is code for.
- **base** (int) – The address `blob` starts at, used to enforce instruction alignment.

**Return type**
Iterator[tuple[int, int, int]]

**Returns**
An iterator of `(start, end, branch_target)` offsets into `blob`. Only preambles whose stack-growth branch goes forward are reported; the caller must still check that `branch_target` lies in the same executable region.

### angr.analyses.cfg.indirect_jump_resolvers

**class** angr.analyses.cfg.indirect_jump_resolvers.**AArch64MachOGotResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT entries on AArch64 MachO binaries.

**__init__**(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

Resolves the GOT entries in AARCH64 Mach-O binaries, where plt stubs are of manner:

```
adrp    x16, #0x100000
ldr     x16, [x16,#0x10]
br      x16
```

**Parameters**
**func_graph_complete** (*bool*)

**class** angr.analyses.cfg.indirect_jump_resolvers.**AMD64ElfGotResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT entries on AMD64 ELF binaries.

**__init__**(*project*)

**class** angr.analyses.cfg.indirect_jump_resolvers.**AMD64PeIatResolver**

Bases: *IndirectJumpResolver*

A timeless indirect call/jump resolver for IAT in amd64 PEs.

**__init__**(*project*)

class angr.analyses.cfg.indirect_jump_resolvers.**ArmElfFastResolver**

Bases: *IndirectJumpResolver*

Resolves indirect jumps in ARM ELF binaries

**__init__**(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

The main resolving function.

**Parameters**

- **cfg** – A CFG instance.
- **addr** (*int*) – Address of the IRSB.
- **func_addr** (*int*) – Address of the function.
- **block** – The IRSB.
- **jumpkind** (*str*) – The jumpkind.
- **func_graph_complete** (*bool*)

**Returns**

**Return type**
tuple

class angr.analyses.cfg.indirect_jump_resolvers.**ConstantResolver**

Bases: *IndirectJumpResolver*

Resolve an indirect jump by running a constant propagation on the entire function and check if the indirect jump can be resolved to a constant value. This resolver must be run after all other more specific resolvers.

**__init__**(*project*, *max_func_nodes=512*)

**Parameters**
**max_func_nodes** (*int*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

This function does the actual resolve. Our process is easy: Propagate all values inside the function specified, then extract the tmp_var used for the indirect jump from the basic block. Use the tmp var to locate the constant value stored in the replacements. If not present, returns False tuple.

**Parameters**

- **cfg** – CFG with specified function
- **addr** (int) – Address of indirect jump
- **func_addr** (int) – Address of function of indirect jump
- **block** (*Block*) – Block of indirect jump (Block object)
- **jumpkind** (str) – VEX jumpkind (Ijk_Boring or Ijk_Call)
- **func_graph_complete** (*bool*)

**Returns**
Bool tuple with replacement address

**class** angr.analyses.cfg.indirect_jump_resolvers.**JumpTableResolver**

Bases: *IndirectJumpResolver*

A generic jump table resolver.

**This is a fast jump table resolution. For performance concerns, we made the following assumptions:**

- The final jump target comes from the memory.
- The final jump target must be directly read out of the memory, without any further modification or altering.

Progressively larger program slices will be analyzed to determine jump table location and size. If the size of the table cannot be determined, a *guess* will be made based on how many entries in the table *appear* valid.

**__init__**(*project*, *resolve_calls=True*)

**Parameters**
**resolve_calls** (*bool*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

Resolves jump tables.

**Parameters**

- **cfg** – A CFG instance.
- **addr** (*int*) – IRSB address.
- **func_addr** (*int*) – The function address.
- **block** (*pyvex.IRSB*) – The IRSB.
- **func_graph_complete** (*bool*)

**Returns**
A bool indicating whether the indirect jump is resolved successfully, and a list of resolved targets

**Return type**
tuple

**class** angr.analyses.cfg.indirect_jump_resolvers.**MemoryLoadResolver**

Bases: *IndirectJumpResolver*

Resolve an indirect jump that looks like the following:

```
.text:
                call    off_3314A8

.data:
off_3314A8      dd offset sub_1E426F
```

This indirect jump resolver may not be the best solution for all cases (e.g., when the .data section can be intentionally altered by the binary itself).

**__init__**(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

**Parameters**

- **cfg** – CFG with specified function

- **addr** (int) – Address of indirect jump
- **func_addr** (int) – Address of function of indirect jump
- **block** (IRSB) – Block of indirect jump (Block object)
- **jumpkind** (str) – VEX jumpkind (Ijk_Boring or Ijk_Call)
- **func_graph_complete** (*bool*)

**Returns**
Bool tuple with replacement address

**class** angr.analyses.cfg.indirect_jump_resolvers.**MipsElfFastResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for R9-based indirect function calls in MIPS ELFs.

**__init__**(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

Wrapper for _resolve that slowly increments the max_depth used by Blade for finding sources until we can resolve the addr or we reach the default max_depth

**Parameters**

- **cfg** – A CFG instance.
- **addr** (*int*) – IRSB address.
- **func_addr** (*int*) – The function address.
- **block** (*pyvex.IRSB*) – The IRSB.
- **jumpkind** (*str*) – The jumpkind.
- **func_graph_complete** (*bool*)

**Returns**
If it was resolved and targets alongside it

**Return type**
tuple

**class** angr.analyses.cfg.indirect_jump_resolvers.**MipsElfGotResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT stub entries in MIPS ELF binaries.

Reference: MIPS Assembly Language Programmer's Guide, Calling Position Independent Functions

**__init__**(*project*)

**class** angr.analyses.cfg.indirect_jump_resolvers.**SyscallResolver**

Bases: *IndirectJumpResolver*

Resolve syscalls to SimProcedures.

**__init__**(*project*)

**class** angr.analyses.cfg.indirect_jump_resolvers.**X86ElfPicPltResolver**

Bases: *IndirectJumpResolver*

In X86 ELF position-independent code, PLT stubs uses ebx to resolve library calls, where ebx stores the address to the beginning of the GOT. We resolve the target by forcing ebx to be the beginning of the GOT and simulate the execution in fast path mode.

__init__(*project*)

class angr.analyses.cfg.indirect_jump_resolvers.**X86PeIatResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for IAT in x86 PEs and xbes.

__init__(*project*)

### Submodules


*aarch64_macho_got*
*amd64_elf_got*
*amd64_pe_iat*
*arm_elf_fast*
*const_resolver*
*constant_value_manager*
*default_resolvers*
*jumptable*
*memload_resolver*
*mips_elf_fast*
*mips_elf_got*
*propagator_utils*
*resolver*
*syscall_resolver*
*x86_elf_pic_plt*
*x86_pe_iat*


### angr.analyses.cfg.indirect_jump_resolvers.aarch64_macho_got

class angr.analyses.cfg.indirect_jump_resolvers.aarch64_macho_got.**AArch64MachOGotResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT entries on AArch64 MachO binaries.

__init__(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

Resolves the GOT entries in AARCH64 Mach-O binaries, where plt stubs are of manner:

```
adrp    x16, #0x100000
ldr     x16, [x16,#0x10]
br      x16
```

**Parameters**

**func_graph_complete** (*bool*)

### angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got

class angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.**AMD64ElfGotResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT entries on AMD64 ELF binaries.

__init__(*project*)

### angr.analyses.cfg.indirect_jump_resolvers.amd64_pe_iat

class angr.analyses.cfg.indirect_jump_resolvers.amd64_pe_iat.**AMD64PeIatResolver**

Bases: *IndirectJumpResolver*

A timeless indirect call/jump resolver for IAT in amd64 PEs.

**__init__**(*project*)

### angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast

class angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.**ArmElfFastResolver**

Bases: *IndirectJumpResolver*

Resolves indirect jumps in ARM ELF binaries

**__init__**(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

The main resolving function.

**Parameters**

- **cfg** – A CFG instance.
- **addr** (*int*) – Address of the IRSB.
- **func_addr** (*int*) – Address of the function.
- **block** – The IRSB.
- **jumpkind** (*str*) – The jumpkind.
- **func_graph_complete** (*bool*)

**Returns**

**Return type**
tuple

### angr.analyses.cfg.indirect_jump_resolvers.const_resolver

angr.analyses.cfg.indirect_jump_resolvers.const_resolver.**exists_in_replacements**(*replacements*, *block_loc*, *tmp_var*)

class angr.analyses.cfg.indirect_jump_resolvers.const_resolver.**ConstantResolver**

Bases: *IndirectJumpResolver*

Resolve an indirect jump by running a constant propagation on the entire function and check if the indirect jump can be resolved to a constant value. This resolver must be run after all other more specific resolvers.

**__init__**(*project*, *max_func_nodes=512*)

**Parameters**
**max_func_nodes** (*int*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

This function does the actual resolve. Our process is easy: Propagate all values inside the function specified, then extract the tmp_var used for the indirect jump from the basic block. Use the tmp var to locate the constant value stored in the replacements. If not present, returns False tuple.

**Parameters**

- **cfg** – CFG with specified function
- **addr** (int) – Address of indirect jump
- **func_addr** (int) – Address of function of indirect jump
- **block** (*Block*) – Block of indirect jump (Block object)
- **jumpkind** (str) – VEX jumpkind (Ijk_Boring or Ijk_Call)
- **func_graph_complete** (*bool*)

**Returns**
Bool tuple with replacement address

### angr.analyses.cfg.indirect_jump_resolvers.constant_value_manager

**class** angr.analyses.cfg.indirect_jump_resolvers.constant_value_manager.**ConstantValueManager**

Bases: object

Manages the loading of registers who hold constant values.

**__init__**(*project*, *kb*, *func*, *ij_addr*)

**Parameters**

- **project** (Project)
- **func** (Function)
- **ij_addr** (*int*)

**project**

**kb**

**func**

**indirect_jump_addr**

**mapping:** dict[Any, dict[Any, Base]] | None

**reg_read_callback**(*state*)

**Parameters**
**state** (SimState)

### angr.analyses.cfg.indirect_jump_resolvers.default_resolvers

angr.analyses.cfg.indirect_jump_resolvers.default_resolvers.**default_indirect_jump_resolvers**(*obj*, *project*)

### angr.analyses.cfg.indirect_jump_resolvers.jumptable

**exception** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**NotAJumpTableNotification**

Bases: *AngrError*

Exception raised to indicate this is not (or does not appear to be) a jump table.

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**UninitReadMeta**

Bases: object

Uninitialized read remapping details.

**uninit_read_base = 201326592**

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**AddressTransformationTypes**

Bases: int, Enum

Address transformation operations.

**Assignment = 0**

**SignedExtension = 1**

**UnsignedExtension = 2**

**Truncation = 3**

**Or1 = 4**

**ShiftLeft = 5**

**ShiftRight = 6**

**Add = 7**

**Load = 8**

**__new__**(*value*)

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**AddressTransformation**

Bases: object

Describe and record an address transformation operation.

**__init__**(*op*, *operands*, *first_load=False*)

**Parameters**

- **op** (AddressTransformationTypes)
- **operands** (*list*)
- **first_load** (*bool*)

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**AddressOperand**

Bases: object

The class for the singleton class AddressSingleton. It represents the address being transformed before using as an indirect jump target.

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**Tmp**

Bases: object

For modeling Tmp variables.

**__init__**(*tmp_idx*)

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**JumpTargetBaseAddr**

Bases: object

Model for jump targets and their data origin.

**__init__**(*stmt_loc*, *stmt*, *tmp*, *base_addr=None*, *tmp_1=None*)

**Parameters**

- **stmt** (*IRStmt*)
- **tmp** (*int*)
- **base_addr** (*int | None*)
- **tmp_1** (*int | None*)

**property base_addr_available**

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**JumpTableProcessorState**

Bases: object

The state used in JumpTableProcessor.

**__init__**(*arch*)

**arch**

**is_jumptable: bool | None**

**stmts_to_instrument**

**regs_to_initialize**

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**RegOffsetAnnotation**

Bases: Annotation

Register Offset annotation.

**relocatable = False**

**eliminatable = False**

**__init__**(*reg_offset*)

**Parameters**

**reg_offset** (RegisterOffset)

**reg_offset**

class angr.analyses.cfg.indirect_jump_resolvers.jumptable.**JumpTableProcessor**

Bases: *SimEngineNostmtVEX*[*JumpTableProcessorState*, BV, *JumpTableProcessorState*], *ClaripyDataVEXEngineMixin*[*JumpTableProcessorState*, BV, *JumpTableProcessorState*, None]

Implements a simple and stupid data dependency tracking for stack and register variables.

Also determines which statements to instrument during static execution of the slice later. For example, the following example is not uncommon in non-optimized binaries:

```
    mov  [rbp+var_54], 1
loc_4051a6:
    cmp  [rbp+var_54], 6
    ja   loc_405412 (default)
loc_4051b0:
    mov  eax, [rbp+var_54]
    mov  rax, qword [rax*8+0x223a01]
    jmp  rax
```

We want to instrument the first instruction and replace the constant 1 with a symbolic variable, otherwise we will not be able to recover all jump targets later in block 0x4051b0.

**__init__**(*project*, *indirect_jump_node_pred_addrs*, *bp_sp_diff=256*)

**Parameters**
**indirect_jump_node_pred_addrs** (*set[int]*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**StoreHook**

Bases: object

Hook for memory stores.

**static hook**(*state*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**LoadHook**

Bases: object

Hook for memory loads.

**__init__**()

**hook_before**(*state*)

**hook_after**(*state*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**PutHook**

Bases: object

Hook for register writes.

**static hook**(*state*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**ShrBoundsConstraintHook**

Bases: object

Hook for rewriting folded bound-check constraints.

libVEX 3.27+ spechelpers fold the unsigned bound check *x <=u 2**k-1* into *Shr(x, k) == 0*. The VSA constraint handler keys its replacement on the LShR expression instead of x itself, so the bound never narrows x (and thus the jump table address) during slice execution. Rewrite such constraints into the canonical ULE/UGT form.

**static hook**(*state*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**RegisterInitializerHook**

Bases: object

Hook for register init.

**__init__**(*reg_offset*, *reg_bits*, *initial_value*)

**hook**(*state*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**BSSHook**

Bases: object

Hook for BSS read/write.

**__init__**(*project*, *bss_regions*)

**bss_memory_read_hook**(*state*)

**bss_memory_write_hook**(*state*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**MIPSGPHook**

Bases: object

Hooks all reads from and writes into the gp register for MIPS32 binaries.

**__init__**(*gp_offset*, *gp*)

**Parameters**

- **gp_offset** (*int*)
- **gp** (*int*)

**gp_register_read_hook**(*state*)

**gp_register_write_hook**(*state*)

**class** angr.analyses.cfg.indirect_jump_resolvers.jumptable.**JumpTableResolver**

Bases: *IndirectJumpResolver*

A generic jump table resolver.

**This is a fast jump table resolution. For performance concerns, we made the following assumptions:**

- The final jump target comes from the memory.
- The final jump target must be directly read out of the memory, without any further modification or altering.

Progressively larger program slices will be analyzed to determine jump table location and size. If the size of the table cannot be determined, a *guess* will be made based on how many entries in the table *appear* valid.

**__init__**(*project*, *resolve_calls=True*)

**Parameters**
**resolve_calls** (*bool*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

Resolves jump tables.

**Parameters**

- **cfg** – A CFG instance.
- **addr** (*int*) – IRSB address.
- **func_addr** (*int*) – The function address.
- **block** (*pyvex.IRSB*) – The IRSB.
- **func_graph_complete** (*bool*)

**Returns**
A bool indicating whether the indirect jump is resolved successfully, and a list of resolved targets

**Return type**
tuple

### angr.analyses.cfg.indirect_jump_resolvers.memload_resolver

**class** angr.analyses.cfg.indirect_jump_resolvers.memload_resolver.**MemoryLoadResolver**

Bases: *IndirectJumpResolver*

Resolve an indirect jump that looks like the following:

```
.text:
                call    off_3314A8

.data:
off_3314A8      dd offset sub_1E426F
```

This indirect jump resolver may not be the best solution for all cases (e.g., when the .data section can be intentionally altered by the binary itself).

**__init__**(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

**Parameters**

- **cfg** – CFG with specified function
- **addr** (int) – Address of indirect jump
- **func_addr** (int) – Address of function of indirect jump
- **block** (IRSB) – Block of indirect jump (Block object)
- **jumpkind** (str) – VEX jumpkind (Ijk_Boring or Ijk_Call)
- **func_graph_complete** (*bool*)

**Returns**
Bool tuple with replacement address

### angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast

angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.**enable_profiling**()

angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.**disable_profiling**()

**class** angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.**Case2Result**

Bases: Enum

Describes the result of resolving case 2 function calls.

**SUCCESS = 0**

**FAILURE = 1**

**RESUME = 2**

class angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.**MipsElfFastResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for R9-based indirect function calls in MIPS ELFs.

**__init__**(*project*)

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

Wrapper for _resolve that slowly increments the max_depth used by Blade for finding sources until we can resolve the addr or we reach the default max_depth

**Parameters**

- **cfg** – A CFG instance.
- **addr** (*int*) – IRSB address.
- **func_addr** (*int*) – The function address.
- **block** (*pyvex.IRSB*) – The IRSB.
- **jumpkind** (*str*) – The jumpkind.
- **func_graph_complete** (*bool*)

**Returns**
If it was resolved and targets alongside it

**Return type**
tuple

### angr.analyses.cfg.indirect_jump_resolvers.mips_elf_got

class angr.analyses.cfg.indirect_jump_resolvers.mips_elf_got.**MipsElfGotResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver that resolves GOT stub entries in MIPS ELF binaries.

Reference: MIPS Assembly Language Programmer's Guide, Calling Position Independent Functions

**__init__**(*project*)

### angr.analyses.cfg.indirect_jump_resolvers.propagator_utils

class angr.analyses.cfg.indirect_jump_resolvers.propagator_utils.**PropagatorLoadCallback**

Bases: object

Implement the load callback for Propagator that should be used across all jump table resolvers.

**__init__**(*project*)

**propagator_load_callback**(*addr*, *size*)

**Return type**
bool

**Parameters**

- **addr** (*BV | int*)
- **size** (*int*)

### angr.analyses.cfg.indirect_jump_resolvers.resolver

**class** angr.analyses.cfg.indirect_jump_resolvers.resolver.**IndirectJumpResolver**

Bases: object

**__init__**(*project*, *timeless=False*, *base_state=None*)

**filter**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*)

Check if this resolution method may be able to resolve the indirect jump or not.

**Parameters**

- **addr** (*int*) – Basic block address of this indirect jump.
- **func_addr** (*int*) – Address of the function that this indirect jump belongs to.
- **block** – The basic block. The type is determined by the backend being used. It's pyvex.IRSB if pyvex is used as the backend.
- **jumpkind** (*str*) – The jumpkind.

**Returns**

True if it is possible for this resolution method to resolve the specific indirect jump, False otherwise.

**Return type**

bool

**resolve**(*cfg*, *addr*, *func_addr*, *block*, *jumpkind*, *func_graph_complete=True*, ***kwargs*)

Resolve an indirect jump.

**Parameters**

- **cfg** – The CFG analysis object.
- **addr** (*int*) – Basic block address of this indirect jump.
- **func_addr** (*int*) – Address of the function that this indirect jump belongs to.
- **block** – The basic block. The type is determined by the backend being used. It's pyvex.IRSB if pyvex is used as the backend.
- **jumpkind** (*str*) – The jumpkind.
- **func_graph_complete** (bool) – True if the function graph is complete at this point (except for nodes that this indirect jump node dominates).

**Returns**

A tuple of a boolean indicating whether the resolution is successful or not, and a list of resolved targets (ints).

**Return type**

tuple

### angr.analyses.cfg.indirect_jump_resolvers.syscall_resolver

**class** angr.analyses.cfg.indirect_jump_resolvers.syscall_resolver.**SyscallResolver**

Bases: *IndirectJumpResolver*

Resolve syscalls to SimProcedures.

**__init__**(*project*)

### angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt

**class** angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt.**X86ElfPicPltResolver**

Bases: *IndirectJumpResolver*

In X86 ELF position-independent code, PLT stubs uses ebx to resolve library calls, where ebx stores the address to the beginning of the GOT. We resolve the target by forcing ebx to be the beginning of the GOT and simulate the execution in fast path mode.

**__init__**(*project*)

### angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat

**class** angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat.**X86PeIatResolver**

Bases: *IndirectJumpResolver*

A timeless indirect jump resolver for IAT in x86 PEs and xbes.

**__init__**(*project*)

### angr.analyses.cfg.meta_structs

angr.analyses.cfg.meta_structs.**get_data_regions_from_meta_regions**(*loader*)

Extract (addr, size, sort) tuples for all metadata data regions across all loaded objects.

Flattens DataDirectory sub-regions so each entry maps to a contiguous memory range suitable for marking in CFGFast's _seg_list.

**Return type**
list[tuple[int, int, str | None]]

**Parameters**
**loader** (*Loader*)

angr.analyses.cfg.meta_structs.**get_function_hints_from_meta_regions**(*loader*)

Extract (addr, name) pairs for functions discovered from metadata.

Uses meta_function_hints populated by backends (e.g., PE export table entries).

**Return type**
list[tuple[int, str | None]]

**Parameters**
**loader** (*Loader*)

angr.analyses.cfg.meta_structs.**get_pointer_array_hints**(*loader*)

Extract pointer array hints and return tuples of (addr, byte count) for each pointer array.

**Return type**
list[tuple[int, int]]

**Parameters**
**loader** (*Loader*)

angr.analyses.cfg.meta_structs.**get_pointer_array_hints_pe**(*pe*)

Extract pointer array hints from a PE object and return tuples of (addr, byte count) for each pointer array.

**Return type**
list[tuple[int, int]]

**Parameters**
**pe** (*PE*)

### angr.analyses.cfg.pe_msvc_eh_structs

Struct definitions and parsing utilities for MSVC C++ exception handling structures found in x86 PE binaries.

References: - ___CxxFrameHandler3: The MSVC C++ exception handler for 32-bit binaries. - __EH_prolog3 / __EH_prolog3_catch / __EH_prolog3_GS: C++ EH prolog helpers. - __SEH_prolog4 / __SEH_prolog4_GS: SEH prolog helpers. - FuncInfo: Describes exception handling metadata for a function. - UnwindMapEntry: Describes state unwinding actions during exception handling.

**class** angr.analyses.cfg.pe_msvc_eh_structs.**FuncInfo**

Bases: object

Parsed FuncInfo struct from a 32-bit PE binary.

**__init__**(*addr*, *magic_number*, *max_state*, *p_unwind_map*, *n_try_blocks*, *p_try_block_map*, *n_ip_map_entries*, *p_ip_to_state_map*, *p_es_type_list*, *eh_flags*)

**Parameters**

- **addr** (*int*)
- **magic_number** (*int*)
- **max_state** (*int*)
- **p_unwind_map** (*int*)
- **n_try_blocks** (*int*)
- **p_try_block_map** (*int*)
- **n_ip_map_entries** (*int*)
- **p_ip_to_state_map** (*int*)
- **p_es_type_list** (*int*)
- **eh_flags** (*int*)

**addr**

**magic_number**

**max_state**

**p_unwind_map**

**n_try_blocks**

**p_try_block_map**

**n_ip_map_entries**

**p_ip_to_state_map**

**p_es_type_list**

**eh_flags**

class angr.analyses.cfg.pe_msvc_eh_structs.**UnwindMapEntry**

Bases: object

Parsed UnwindMapEntry struct from a 32-bit PE binary.

**__init__**(*addr*, *to_state*, *action*)

**Parameters**

- **addr** (*int*)
- **to_state** (*int*)
- **action** (*int*)

**addr**

**to_state**

**action**

angr.analyses.cfg.pe_msvc_eh_structs.**parse_funcinfo**(*memory*, *addr*)

Parse a FuncInfo struct at the given address.

**Parameters**

- **memory** – The loader memory interface (supports .load(addr, size)).
- **addr** (int) – The virtual address of the FuncInfo struct.

**Return type**
*FuncInfo* | None

**Returns**
A FuncInfo object, or None if parsing fails.

class angr.analyses.cfg.pe_msvc_eh_structs.**EH4ScopeTable**

Bases: object

Parsed _EH4_SCOPETABLE struct from a 32-bit PE binary.

**__init__**(*addr*, *gs_cookie_offset*, *gs_cookie_xor_offset*, *eh_cookie_offset*, *eh_cookie_xor_offset*, *records*)

**Parameters**

- **addr** (*int*)
- **gs_cookie_offset** (*int*)
- **gs_cookie_xor_offset** (*int*)
- **eh_cookie_offset** (*int*)
- **eh_cookie_xor_offset** (*int*)
- **records** (*list*[EH4ScopeRecord])

**addr**

**gs_cookie_offset**

**gs_cookie_xor_offset**

**eh_cookie_offset**

**eh_cookie_xor_offset**

**records**

**property total_size: int**

class angr.analyses.cfg.pe_msvc_eh_structs.**EH4ScopeRecord**

Bases: object

Parsed _EH4_SCOPETABLE_RECORD struct from a 32-bit PE binary.

**__init__**(*enclosing_level*, *filter_func*, *handler_func*)

**Parameters**

- **enclosing_level** (*int*)
- **filter_func** (*int*)
- **handler_func** (*int*)

**enclosing_level**

**filter_func**

**handler_func**

angr.analyses.cfg.pe_msvc_eh_structs.**parse_eh4_scopetable**(*memory*, *addr*, *code_range=None*)

Parse an _EH4_SCOPETABLE at the given address.

**Parameters**

- **memory** – The loader memory interface.
- **addr** (int) – The virtual address of the _EH4_SCOPETABLE.
- **code_range** (tuple[int, int] | None) – Optional (min_addr, max_addr) of executable memory. When provided, FilterFunc and HandlerFunc pointers are validated against this range.

**Return type**
*EH4ScopeTable* | None

**Returns**
An EH4ScopeTable object, or None if parsing fails.

angr.analyses.cfg.pe_msvc_eh_structs.**parse_unwind_map**(*memory*, *addr*, *count*)

Parse an array of UnwindMapEntry structs.

**Parameters**

- **memory** – The loader memory interface.
- **addr** (int) – The virtual address of the first UnwindMapEntry.
- **count** (int) – The number of entries (maxState from FuncInfo).

**Return type**
list[*UnwindMapEntry*]

**Returns**
A list of UnwindMapEntry objects.

class angr.analyses.cfg.pe_msvc_eh_structs.**TryBlockMapEntry**

Bases: object

Parsed TryBlockMapEntry struct from a 32-bit PE binary.

**__init__**(*addr*, *try_low*, *try_high*, *catch_high*, *n_catches*, *p_handler_array*, *handlers*)

**Parameters**

- **addr** (*int*)
- **try_low** (*int*)
- **try_high** (*int*)
- **catch_high** (*int*)
- **n_catches** (*int*)
- **p_handler_array** (*int*)
- **handlers** (*list[HandlerType]*)

**addr**

**try_low**

**try_high**

**catch_high**

**n_catches**

**p_handler_array**

**handlers**

class angr.analyses.cfg.pe_msvc_eh_structs.**HandlerType**

Bases: object

Parsed HandlerType struct from a 32-bit PE binary.

**__init__**(*addr*, *adjectives*, *p_type*, *disp_catch_obj*, *address_of_handler*)

**Parameters**

- **addr** (*int*)
- **adjectives** (*int*)
- **p_type** (*int*)
- **disp_catch_obj** (*int*)
- **address_of_handler** (*int*)

**addr**

**adjectives**

**p_type**

**disp_catch_obj**

**address_of_handler**

angr.analyses.cfg.pe_msvc_eh_structs.**parse_handler_array**(*memory*, *addr*, *count*)

Parse an array of HandlerType structs.

**Parameters**

- **memory** – The loader memory interface.
- **addr** (int) – The virtual address of the first HandlerType.
- **count** (int) – The number of entries (nCatches from TryBlockMapEntry).

**Return type**
list[*HandlerType*]

**Returns**
A list of HandlerType objects.

angr.analyses.cfg.pe_msvc_eh_structs.**parse_try_block_map**(*memory*, *addr*, *count*)

Parse an array of TryBlockMapEntry structs, including their nested HandlerType arrays.

**Parameters**

- **memory** – The loader memory interface.
- **addr** (int) – The virtual address of the first TryBlockMapEntry.
- **count** (int) – The number of entries (nTryBlocks from FuncInfo).

**Return type**
list[*TryBlockMapEntry*]

**Returns**
A list of TryBlockMapEntry objects.

### 10.3.10 angr.analyses.cfg_slice_to_sink

**class** angr.analyses.cfg_slice_to_sink.**CFGSliceToSink**

Bases: object

The representation of a slice of a CFG.

**__init__**(*target*, *transitions=None*)

**Parameters**

- **target** (angr.knowledge_plugins.functions.function.Function) – The targeted sink, to which every path in the slice leads.
- **transitions** (*Dict[int,List[int]]*) – A mapping representing transitions in the graph. Indexes are source addresses and values a list of destination addresses, for which there exists a transition in the slice from source to destination.

**property transitions**

The transitions in the slice.

**Type**
return Dict[int,List[int]]

**property transitions_as_tuples**

The list of transitions as pairs of (source, destination).

**Type**
return List[Tuple[int,int]]

**property target**

return angr.knowledge_plugins.functions.function.Function: The targeted sink function, from which the slice is constructed.

**property nodes: list[int]**

The complete list of addresses present in the slice.

**Type**
return

**property entrypoints**

Entrypoints are all source addresses that are not the destination address of any transition.

**Return List[int]**
The list of entrypoints addresses.

**add_transitions**(*transitions*)

Add the given transitions to the current slice.

**Parameters**
**transitions** (*Dict[int,List[int]]*) – The list of transitions to be added to *self.transitions*.

**Return Dict[int,List[int]]**
Return the updated list of transitions.

**is_empty**()

Test if a given slice does not contain any transition.

**Return bool**
True if the <CFGSliceToSink> instance does not contain any transitions. False otherwise.

**path_between**(*source*, *destination*, *visited=None*)

Check the existence of a path in the slice between two given node addresses.

**Parameters**

- **source** (int) – The source address.
- **destination** (int) – The destination address.
- **visited** (set[Any] | None) – Used to avoid infinite recursion if loops are present in the slice.

**Return type**
bool

**Returns**
True if there is a path between the source and the destination in the CFG, False if not, or if we have been unable to decide (because of loops).

angr.analyses.cfg_slice_to_sink.**slice_callgraph**(*callgraph*, *cfg_slice_to_sink*)

Slice a callgraph, keeping only the nodes present in the <CFGSliceToSink> representation, and th transitions for which a path exists.

*Note* that this function mutates the graph passed as an argument.

**Parameters**

- **callgraph** (*networkx.MultiDiGraph*) – The callgraph to update.
- **cfg_slice_to_sink** (CFGSliceToSink) – The representation of the slice, containing the data to update the callgraph from.

angr.analyses.cfg_slice_to_sink.**slice_cfg_graph**(*graph*, *cfg_slice_to_sink*)

Slice a CFG graph, keeping only the transitions and nodes present in the <CFGSliceToSink> representation.

*Note* that this function mutates the graph passed as an argument.

**Parameters**

- **graph** (*networkx.DiGraph*) – The graph to slice.
- **cfg_slice_to_sink** (CFGSliceToSink) – The representation of the slice, containing the data to update the CFG from.

**Return networkx.DiGraph**
The sliced graph.

angr.analyses.cfg_slice_to_sink.**slice_function_graph**(*function_graph*, *cfg_slice_to_sink*)

Slice a function graph, keeping only the nodes present in the <CFGSliceToSink> representation.

Because the <CFGSliceToSink> is build from the CFG, and the function graph is *NOT* a subgraph of the CFG, edges of the function graph will no be present in the <CFGSliceToSink> transitions. However, we use the fact that if there is an edge between two nodes in the function graph, then there must exist a path between these two nodes in the slice; Proof idea: - The <CFGSliceToSink> is backward and recursively constructed; - If a node is in the slice, then all its predecessors will be (transitively); - If there is an edge between two nodes in the function graph, there is a path between them in the CFG; - So: The origin node is a transitive predecessor of the destination one, hence if destination is in the slice, then origin will be too.

In consequence, in the end, removing the only nodes not present in the slice, and their related transitions gives us the expected result: a function graph representing (a higher view of) the flow in the slice.

*Note* that this function mutates the graph passed as an argument.

**Parameters**

- **graph** (*networkx.DiGraph*) – The graph to slice.
- **cfg_slice_to_sink** (CFGSliceToSink) – The representation of the slice, containing the data to update the CFG from.

**Return networkx.DiGraph**
The sliced graph.

## Submodules

| | |
|---|---|
| *cfg_slice_to_sink* | |
| *graph* | |
| *transitions* | Some utilitary functions to manage our representation of transitions: |

### angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink

**class** angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.**CFGSliceToSink**

Bases: object

The representation of a slice of a CFG.

**__init__**(*target*, *transitions=None*)

**Parameters**

- **target** (angr.knowledge_plugins.functions.function.Function) – The targeted sink, to which every path in the slice leads.

- **transitions** (*Dict[int,List[int]]*) – A mapping representing transitions in the graph. Indexes are source addresses and values a list of destination addresses, for which there exists a transition in the slice from source to destination.

**property transitions**

The transitions in the slice.

**Type**
return Dict[int,List[int]]

**property transitions_as_tuples**

The list of transitions as pairs of (source, destination).

**Type**
return List[Tuple[int,int]]

**property target**

return angr.knowledge_plugins.functions.function.Function: The targeted sink function, from which the slice is constructed.

**property nodes: list[int]**

The complete list of addresses present in the slice.

**Type**
return

**property entrypoints**

Entrypoints are all source addresses that are not the destination address of any transition.

**Return List[int]**
The list of entrypoints addresses.

**add_transitions**(*transitions*)

Add the given transitions to the current slice.

**Parameters**
**transitions** (*Dict[int,List[int]]*) – The list of transitions to be added to *self.transitions*.

**Return Dict[int,List[int]]**
Return the updated list of transitions.

**is_empty**()

Test if a given slice does not contain any transition.

**Return bool**
True if the <CFGSliceToSink> instance does not contain any transitions. False otherwise.

**path_between**(*source*, *destination*, *visited=None*)

Check the existence of a path in the slice between two given node addresses.

**Parameters**

- **source** (int) – The source address.
- **destination** (int) – The destination address.
- **visited** (set[Any] | None) – Used to avoid infinite recursion if loops are present in the slice.

**Return type**
bool

**Returns**
True if there is a path between the source and the destination in the CFG, False if not, or if we have been unable to decide (because of loops).

### angr.analyses.cfg_slice_to_sink.graph

angr.analyses.cfg_slice_to_sink.graph.**slice_callgraph**(*callgraph*, *cfg_slice_to_sink*)

Slice a callgraph, keeping only the nodes present in the <CFGSliceToSink> representation, and th transitions for which a path exists.

*Note* that this function mutates the graph passed as an argument.

**Parameters**

- **callgraph** (*networkx.MultiDiGraph*) – The callgraph to update.
- **cfg_slice_to_sink** (CFGSliceToSink) – The representation of the slice, containing the data to update the callgraph from.

angr.analyses.cfg_slice_to_sink.graph.**slice_cfg_graph**(*graph*, *cfg_slice_to_sink*)

Slice a CFG graph, keeping only the transitions and nodes present in the <CFGSliceToSink> representation.

*Note* that this function mutates the graph passed as an argument.

**Parameters**

- **graph** (*networkx.DiGraph*) – The graph to slice.
- **cfg_slice_to_sink** (CFGSliceToSink) – The representation of the slice, containing the data to update the CFG from.

**Return networkx.DiGraph**
The sliced graph.

angr.analyses.cfg_slice_to_sink.graph.**slice_function_graph**(*function_graph*, *cfg_slice_to_sink*)

Slice a function graph, keeping only the nodes present in the <CFGSliceToSink> representation.

Because the <CFGSliceToSink> is build from the CFG, and the function graph is *NOT* a subgraph of the CFG, edges of the function graph will no be present in the <CFGSliceToSink> transitions. However, we use the fact that if there is an edge between two nodes in the function graph, then there must exist a path between these two nodes in the slice; Proof idea: - The <CFGSliceToSink> is backward and recursively constructed; - If a node is in the slice, then all its predecessors will be (transitively); - If there is an edge between two nodes in the function graph, there is a path between them in the CFG; - So: The origin node is a transitive predecessor of the destination one, hence if destination is in the slice, then origin will be too.

In consequence, in the end, removing the only nodes not present in the slice, and their related transitions gives us the expected result: a function graph representing (a higher view of) the flow in the slice.

*Note* that this function mutates the graph passed as an argument.

**Parameters**

- **graph** (*networkx.DiGraph*) – The graph to slice.
- **cfg_slice_to_sink** (CFGSliceToSink) – The representation of the slice, containing the data to update the CFG from.

**Return networkx.DiGraph**
The sliced graph.

#### angr.analyses.cfg_slice_to_sink.transitions

**Some utilitary functions to manage our representation of transitions:**
A dictionary, indexed by int (source addresses), which values are list of ints (target addresses).

angr.analyses.cfg_slice_to_sink.transitions.**merge_transitions**(*transitions*, *existing_transitions*)

Merge two dictionaries of transitions together.

**Parameters**

- **transitions** (*Dict[int,List[int]]*) – Some transitions.
- **existing_transitions** (*Dict[int,List[int]]*) – Other transitions.

**Return Dict[int,List[int]]**
The merge of the two parameters.

### 10.3.11 angr.analyses.class_identifier

**class** angr.analyses.class_identifier.**ClassIdentifier**

Bases: *Analysis*

This is a class identifier for non stripped or partially stripped binaries, it identifies classes based on the demangled function names, and also assigns functions to their respective classes based on their names. It also uses the results from the VtableFinder analysis to assign the corresponding vtable to the classes.

self.classes contains a mapping between class names and SimCppClass objects

e.g. A::tool() and A::qux() belong to the class A

**__init__**()

### 10.3.12 angr.analyses.code_tagging

**class** angr.analyses.code_tagging.**CodeTags**

Bases: object

**HAS_XOR = 'HAS_XOR'**

**HAS_BITSHIFTS = 'HAS_BITSHIFTS'**

**HAS_SQL = 'HAS_SQL'**

**LARGE_SWITCH = 'LARGE_SWITCH'**

**class** angr.analyses.code_tagging.**CodeTagging**

Bases: *Analysis*

**__init__**(*func*)

**analyze**()

**has_xor**()

Detects if there is any xor operation in the function.

**Returns**
Tags

**has_bitshifts**()

Detects if there is any bitwise operation in the function.

**Returns**
Tags.

**has_sql**()

Detects if there is any reference to strings that look like SQL queries.

### 10.3.13 angr.analyses.codecave

**class** angr.analyses.codecave.**CodeCaveClassification**

Bases: Enum

Type of code caves.

**ALIGNMENT = 1**

**UNREACHABLE = 2**

**class** angr.analyses.codecave.**CodeCave**

Bases: object

Describes a code cave in a binary.

**func:** *Function* | None

**addr:** int

**size:** int

**classification:** *CodeCaveClassification*

**__init__**(*func*, *addr*, *size*, *classification*)

**Parameters**

- **func** (Function | *None*)
- **addr** (*int*)
- **size** (*int*)
- **classification** (CodeCaveClassification)

**Return type**
None

**class** angr.analyses.codecave.**CodeCaveAnalysis**

Bases: *Analysis*

Best-effort static location of potential vacant code caves for possible code injection: - Padding functions - Unreachable code

**__init__**()

**codecaves:** list[*CodeCave*]

### 10.3.14 angr.analyses.complete_calling_conventions

**class** angr.analyses.complete_calling_conventions.**CallingConventionAnalysisMode**

Bases: Enum

The mode of calling convention analysis.

FAST: Using FactCollector to collect facts, then use facts for calling convention analysis. FASTISH: Using FactCollector with additional introspection to identify behavior of function arguments. VARIABLES: Using variables in VariableManager for calling convention analysis.

**FAST = 'fast'**

**FASTISH = 'fastish'**

**VARIABLES = 'variables'**

**class** angr.analyses.complete_calling_conventions.**CompleteCallingConventionsAnalysis**

Bases: *Analysis*

Implements full-binary calling convention analysis. During the initial analysis of a binary, you may set *recover_variables* to True so that it will perform variable recovery on each function before performing calling convention analysis.

**__init__**(*mode=CallingConventionAnalysisMode.FASTISH*, *recover_variables=False*, *low_priority=False*, *force=False*, *cfg=None*, *analyze_callsites=False*, *skip_signature_matched_functions=False*, *max_function_blocks=None*, *max_function_size=None*, *workers=0*, *cc_callback=None*, *prioritize_func_addrs=None*, *skip_other_funcs=False*, *auto_start=True*, *func_graphs=None*, *target_functions=None*)

**Parameters**

- **recover_variables** – Recover variables on each function before performing calling convention analysis.
- **low_priority** – Run in the background - periodically release GIL.
- **force** – Perform calling convention analysis on functions even if they have calling conventions or prototypes already specified (or previously recovered).
- **cfg** (*CFGFast* | *CFGModel* | None) – The control flow graph model, which will be passed to CallingConventionAnalysis.
- **analyze_callsites** (bool) – Consider artifacts at call sites when performing calling convention analysis.
- **skip_signature_matched_functions** (bool) – Do not perform calling convention analysis on functions that match against existing FLIRT signatures.
- **max_function_blocks** (int | None) – Do not perform calling convention analysis on functions with more than the specified number of blocks. Setting it to None disables this check.
- **max_function_size** (int | None) – Do not perform calling convention analysis on functions whose sizes are more than *max_function_size*. Setting it to None disables this check.
- **workers** (int) – Number of multiprocessing workers.
- **mode** (CallingConventionAnalysisMode)
- **cc_callback** (*Callable | None*)
- **prioritize_func_addrs** (*list[int] | set[int] | None*)

- **skip_other_funcs** (*bool*)
- **auto_start** (*bool*)
- **func_graphs** (*dict[int, DiGraph] | None*)
- **target_functions** (*set[int] | None*)

**prototype_libnames: set[str]**

**work**()

**prioritize_functions**(*func_addrs_to_prioritize*)

Prioritize the analysis of specified functions.

**Parameters**
**func_addrs_to_prioritize** (Iterable[int]) – A collection of function addresses to analyze first.

**static function_needs_variable_recovery**(*func*)

Check if running variable recovery on the function is the only way to determine the calling convention of the this function.

We do not need to run variable recovery to determine the calling convention of a function if: - The function is a SimProcedure. - The function is a PLT stub. - The function is a library function and we already know its prototype.

**Parameters**
**func** – The function object.

**Returns**
True if we must run VariableRecovery before we can determine what the calling convention of this function is. False otherwise.

**Return type**
bool

### 10.3.15 angr.analyses.congruency_check

**class** angr.analyses.congruency_check.**CongruencyCheck**

Bases: *Analysis*

This is an analysis to ensure that angr executes things identically with different execution backends (i.e., unicorn vs vex).

**__init__**(*throw=False*)

Initializes a CongruencyCheck analysis.

**Parameters**
**throw** – whether to raise an exception if an incongruency is found.

**set_state_options**(*left_add_options=None*, *left_remove_options=None*, *right_add_options=None*, *right_remove_options=None*)

Checks that the specified state options result in the same states over the next *depth* states.

**set_states**(*left_state*, *right_state*)

Checks that the specified paths stay the same over the next *depth* states.

**set_simgr**(*simgr*)

**run**(*depth=None*)

Checks that the paths in the specified path group stay the same over the next *depth* bytes.

The path group should have a "left" and a "right" stash, each with a single path.

**compare_path_group**(*pg*)

**compare_states**(*sl*, *sr*)

Compares two states for similarity.

**compare_paths**(*pl*, *pr*)

### 10.3.16 angr.analyses.data_dep

**class** angr.analyses.data_dep.**BaseDepNode**

Bases: object

Base class for all nodes in a data-dependency graph

**__init__**(*type_*, *sim_act*)

**Parameters**

- **type_** (*int*)
- **sim_act** (SimActionData)

**value_tuple**()

**Return type**
tuple[BV, int]

**Returns**
A tuple containing the node's value as a BV and as an evaluated integer

**property ast: BV**

**property type: int**

Getter :return: An integer defined in DepNodeTypes, represents the subclass type of this DepNode.

**class** angr.analyses.data_dep.**ConstantDepNode**

Bases: *BaseDepNode*

Used to create a DepNode that will hold a constant, numeric value Uniquely identified by its value

**__init__**(*sim_act*, *value*)

**Parameters**

- **sim_act** (SimActionData)
- **value** (*int*)

**class** angr.analyses.data_dep.**DataDependencyGraphAnalysis**

Bases: *Analysis*

This is a DYNAMIC data dependency graph that utilizes a given SimState to produce a DDG graph that is accurate to the path the program took during execution.

This analysis utilizes the SimActionData objects present in the provided SimState's action history to generate the dependency graph.

__init__(*end_state*, *start_from=None*, *end_at=None*, *block_addrs=None*)

**Parameters**

- **end_state** (*SimState*) – Simulation state used to extract all SimActionData
- **start_from** (int | None) – An address or None, Specifies where to start generation of DDG
- **end_at** (int | None) – An address or None, Specifies where to end generation of DDG
- **block_addrs** (list[int] | None) – List of block addresses that the DDG analysis should be run on

property graph: DiGraph | None

property simplified_graph: DiGraph | None

property sub_graph: DiGraph | None

get_data_dep(*g_node*, *include_tmp_nodes*, *backwards*)

**Return type**
DiGraph | None

**Parameters**

- **g_node** (BaseDepNode)
- **include_tmp_nodes** (*bool*)
- **backwards** (*bool*)

class angr.analyses.data_dep.**DepNodeTypes**

Bases: object

Enumeration of types of BaseDepNode supported by this analysis

Memory = 1

Register = 2

Tmp = 3

Constant = 4

class angr.analyses.data_dep.**MemDepNode**

Bases: *BaseDepNode*

Used to represent SimActions of type MEM

__init__(*sim_act*, *addr*)

**Parameters**

- **sim_act** (SimActionData)
- **addr** (*int*)

property width: int

**classmethod cast_to_mem**(*base_dep_node*)

Casts a BaseDepNode into a MemDepNode

**Parameters**
**base_dep_node** (BaseDepNode)

**class** angr.analyses.data_dep.**RegDepNode**

Bases: *VarDepNode*

Base class for representing SimActions of TYPE reg

**__init__**(*sim_act*, *reg*, *arch_name=''*)

**Parameters**

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

**property reg_size:** int

**class** angr.analyses.data_dep.**TmpDepNode**

Bases: *VarDepNode*

Used to represent SimActions of type TMP

**__init__**(*sim_act*, *reg*, *arch_name=''*)

**Parameters**

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

**class** angr.analyses.data_dep.**VarDepNode**

Bases: *BaseDepNode*

Abstract class for representing SimActions of TYPE reg or tmp

**__init__**(*type_*, *sim_act*, *reg*, *arch_name=''*)

**Parameters**

- **type_** (*int*)
- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

**property display_name:** str

## Submodules

| | |
|---|---|
| *data_dependency_analysis* | Defines analysis that will generate a dynamic data-dependency graph |
| *dep_nodes* | |

continues on next page

Table 8 – continued from previous page

| |
|---|
| *sim_act_location* |

### angr.analyses.data_dep.data_dependency_analysis

Defines analysis that will generate a dynamic data-dependency graph

**class** angr.analyses.data_dep.data_dependency_analysis.**NodalAnnotation**

Bases: Annotation

Allows a node to be stored as an annotation to a BV in a DefaultMemory instance

**relocatable = False**

Can not be relocated in a simplification

**eliminatable = False**

Can not be eliminated in a simplification

**__init__**(*node*)

**Parameters**
**node** (BaseDepNode)

**class** angr.analyses.data_dep.data_dependency_analysis.**DataDependencyGraphAnalysis**

Bases: *Analysis*

This is a DYNAMIC data dependency graph that utilizes a given SimState to produce a DDG graph that is accurate to the path the program took during execution.

This analysis utilizes the SimActionData objects present in the provided SimState's action history to generate the dependency graph.

**__init__**(*end_state*, *start_from=None*, *end_at=None*, *block_addrs=None*)

**Parameters**

- **end_state** (*SimState*) – Simulation state used to extract all SimActionData
- **start_from** (int | None) – An address or None, Specifies where to start generation of DDG
- **end_at** (int | None) – An address or None, Specifies where to end generation of DDG
- **block_addrs** (list[int] | None) – List of block addresses that the DDG analysis should be run on

**property graph: DiGraph | None**

**property simplified_graph: DiGraph | None**

**property sub_graph: DiGraph | None**

**get_data_dep**(*g_node*, *include_tmp_nodes*, *backwards*)

**Return type**
DiGraph | None

**Parameters**

- **g_node** (BaseDepNode)
- **include_tmp_nodes** (*bool*)
- **backwards** (*bool*)

### angr.analyses.data_dep.dep_nodes

**class** angr.analyses.data_dep.dep_nodes.**DepNodeTypes**

Bases: object

Enumeration of types of BaseDepNode supported by this analysis

**Memory = 1**

**Register = 2**

**Tmp = 3**

**Constant = 4**

**class** angr.analyses.data_dep.dep_nodes.**BaseDepNode**

Bases: object

Base class for all nodes in a data-dependency graph

**__init__**(*type_*, *sim_act*)

**Parameters**

- **type_** (*int*)
- **sim_act** (SimActionData)

**action_id: int**

**value: int | None**

**value_tuple**()

**Return type**
tuple[BV, int]

**Returns**
A tuple containing the node's value as a BV and as an evaluated integer

**property ast: BV**

**property type: int**

Getter :return: An integer defined in DepNodeTypes, represents the subclass type of this DepNode.

**class** angr.analyses.data_dep.dep_nodes.**ConstantDepNode**

Bases: *BaseDepNode*

Used to create a DepNode that will hold a constant, numeric value Uniquely identified by its value

**__init__**(*sim_act*, *value*)

**Parameters**

- **sim_act** (SimActionData)
- **value** (*int*)

**class** angr.analyses.data_dep.dep_nodes.**MemDepNode**

Bases: *BaseDepNode*

Used to represent SimActions of type MEM

__init__(*sim_act*, *addr*)

**Parameters**

- **sim_act** (SimActionData)
- **addr** (*int*)

property width: int

classmethod cast_to_mem(*base_dep_node*)

Casts a BaseDepNode into a MemDepNode

**Parameters**

**base_dep_node** (BaseDepNode)

class angr.analyses.data_dep.dep_nodes.**VarDepNode**

Bases: *BaseDepNode*

Abstract class for representing SimActions of TYPE reg or tmp

__init__(*type_*, *sim_act*, *reg*, *arch_name=''*)

**Parameters**

- **type_** (*int*)
- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

property display_name: str

class angr.analyses.data_dep.dep_nodes.**TmpDepNode**

Bases: *VarDepNode*

Used to represent SimActions of type TMP

__init__(*sim_act*, *reg*, *arch_name=''*)

**Parameters**

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

class angr.analyses.data_dep.dep_nodes.**RegDepNode**

Bases: *VarDepNode*

Base class for representing SimActions of TYPE reg

__init__(*sim_act*, *reg*, *arch_name=''*)

**Parameters**

- **sim_act** (SimActionData)
- **reg** (*int*)
- **arch_name** (*str*)

property reg_size: int

#### angr.analyses.data_dep.sim_act_location

**class** angr.analyses.data_dep.sim_act_location.**SimActLocation**

Bases: object

Structure-like class used to bundle the instruction address and statement index of a given SimAction in order to uniquely identify a given SimAction

**__init__**(*bbl_addr*, *ins_addr*, *stmt_idx*)

**Parameters**

- **bbl_addr** (*int*)
- **ins_addr** (*int*)
- **stmt_idx** (*int*)

**class** angr.analyses.data_dep.sim_act_location.**ParsedInstruction**

Bases: object

Used by parser to facilitate linking with recent ancestors in an efficient manner

**__init__**(*ins_addr*, *min_stmt_idx*, *max_stmt_idx*)

**Parameters**

- **ins_addr** (*int*)
- **min_stmt_idx** (*int*)
- **max_stmt_idx** (*int*)

### 10.3.17 angr.analyses.datagraph_meta

**exception** angr.analyses.datagraph_meta.**DataGraphError**

Bases: Exception

**class** angr.analyses.datagraph_meta.**DataGraphMeta**

Bases: object

**__init__**()

**get_irsb_at**(*addr*)

**pp**(*imarks=False*)

Pretty print the graph. @imarks determine whether the printed graph represents instructions (coarse grained) for easier navigation, or exact statements.

### 10.3.18 angr.analyses.ddg

**class** angr.analyses.ddg.**AST**

Bases: object

A mini implementation for AST

**__init__**(*op*, **operands*)

class angr.analyses.ddg.**ProgramVariable**

Bases: object

Describes a variable in the program at a specific location.

**Variables**

- **variable** (SimVariable) – The variable.
- **location** (CodeLocation) – Location of the variable.

**__init__**(*variable*, *location*, *initial=False*, *arch=None*)

**property short_repr**

class angr.analyses.ddg.**DDGJob**

Bases: object

**__init__**(*cfg_node*, *call_depth*)

class angr.analyses.ddg.**LiveDefinitions**

Bases: object

A collection of live definitions with some handy interfaces for definition killing and lookups.

**__init__**()

Constructor.

**branch**()

Create a branch of the current live definition collection.

**Returns**
A new LiveDefinition instance.

**Return type**
*angr.analyses.ddg.LiveDefinitions*

**copy**()

Make a hard copy of *self*.

**Returns**
A new LiveDefinition instance.

**Return type**
*angr.analyses.ddg.LiveDefinitions*

**add_def**(*variable*, *location*, *size_threshold=32*)

Add a new definition of variable.

**Parameters**

- **variable** (SimVariable) – The variable being defined.
- **location** (CodeLocation) – Location of the variable being defined.
- **size_threshold** (*int*) – The maximum bytes to consider for the variable.

**Returns**
True if the definition was new, False otherwise

**Return type**
bool

**add_defs**(*variable*, *locations*, *size_threshold=32*)

Add a collection of new definitions of a variable.

**Parameters**

- **variable** (SimVariable) – The variable being defined.
- **locations** (*iterable*) – A collection of locations where the variable was defined.
- **size_threshold** (*int*) – The maximum bytes to consider for the variable.

**Returns**
True if any of the definition was new, False otherwise

**Return type**
bool

**kill_def**(*variable*, *location*, *size_threshold=32*)

Add a new definition for variable and kill all previous definitions.

**Parameters**

- **variable** (SimVariable) – The variable to kill.
- **location** (CodeLocation) – The location where this variable is defined.
- **size_threshold** (*int*) – The maximum bytes to consider for the variable.

**Returns**
None

**lookup_defs**(*variable*, *size_threshold=32*)

Find all definitions of the variable.

**Parameters**

- **variable** (SimVariable) – The variable to lookup for.
- **size_threshold** (*int*) – The maximum bytes to consider for the variable. For example, if the variable is 100 byte long, only the first *size_threshold* bytes are considered.

**Returns**
A set of code locations where the variable is defined.

**Return type**
set

**items**()

An iterator that returns all live definitions.

**Returns**
The iterator.

**Return type**
iter

**itervariables**()

An iterator that returns all live variables.

**Returns**
The iterator.

**Return type**
iter

class angr.analyses.ddg.**DDGViewItem**

Bases: object

**__init__**(*ddg*, *variable*, *simplified=False*)

**property depends_on**

**property dependents**

class angr.analyses.ddg.**DDGViewInstruction**

Bases: object

**__init__**(*cfg*, *ddg*, *insn_addr*, *simplified=False*)

**property definitions: list[*DDGViewItem*]**

Get all definitions located at the current instruction address.

**Returns**
A list of ProgramVariable instances.

class angr.analyses.ddg.**DDGView**

Bases: object

A view of the data dependence graph.

**__init__**(*cfg*, *ddg*, *simplified=False*)

class angr.analyses.ddg.**DDG**

Bases: *Analysis*

This is a fast data dependence graph directly generated from our CFG analysis result. The only reason for its existence is the speed. There is zero guarantee for being sound or accurate. You are supposed to use it only when you want to track the simplest data dependence, and you do not care about soundness or accuracy.

For a better data dependence graph, please consider performing a better static analysis first (like Value-set Analysis), and then construct a dependence graph on top of the analysis result (for example, the VFG in angr).

The DDG is based on a CFG, which should ideally be a CFGEmulated generated with the following options:

- keep_state=True to keep all input states
- state_add_options=angr.options.refs to store memory, register, and temporary value accesses

You may want to consider a high value for context_sensitivity_level as well when generating the CFG.

Also note that since we are using states from CFG, any improvement in analysis performed on CFG (like a points-to analysis) will directly benefit the DDG.

**__init__**(*cfg*, *start=None*, *call_depth=None*, *block_addrs=None*)

**Parameters**

- **cfg** – Control flow graph. Please make sure each node has an associated *state* with it, e.g. by passing the keep_state=True and state_add_options=angr.options.refs arguments to CFGEmulated.
- **start** – An address, Specifies where we start the generation of this data dependence graph.
- **call_depth** – None or integers. A non-negative integer specifies how deep we would like to track in the call tree. None disables call_depth limit.
- **block_addrs** (*iterable or None*) – A collection of block addresses that the DDG analysis should be performed on.

**property graph**

A networkx DiGraph instance representing the dependence relations between statements. :rtype: networkx.DiGraph

**Type**
returns

**property data_graph**

Get the data dependence graph.

**Returns**
A networkx DiGraph instance representing data dependence.

**Return type**
networkx.DiGraph

**property simplified_data_graph**

return:

**property ast_graph**

**pp()**

Pretty printing.

**dbg_repr()**

Representation for debugging.

**get_predecessors**(*code_location*)

Returns all predecessors of the code location.

**Parameters**
**code_location** – A CodeLocation instance.

**Returns**
A list of all predecessors.

**function_dependency_graph**(*func*)

Get a dependency graph for the function *func*.

**Parameters**
**func** – The Function object in CFG.function_manager.

**Returns**
A networkx.DiGraph instance.

**data_sub_graph**(*pv*, *simplified=True*, *killing_edges=False*, *excluding_types=None*)

Get a subgraph from the data graph or the simplified data graph that starts from node pv.

**Parameters**

- **pv** (*ProgramVariable*) – The starting point of the subgraph.
- **simplified** (*bool*) – When True, the simplified data graph is used, otherwise the data graph is used.
- **killing_edges** (*bool*) – Are killing edges included or not.
- **excluding_types** (*iterable*) – Excluding edges whose types are among those excluded types.

**Returns**
A subgraph.

**Return type**
networkx.MultiDiGraph

**find_definitions**(*variable*, *location=None*, *simplified_graph=True*)

Find all definitions of the given variable.

**Parameters**

- **variable** (SimVariable)
- **simplified_graph** (*bool*) – True if you just want to search in the simplified graph instead of the normal graph. Usually the simplified graph suffices for finding definitions of register or memory variables.

**Returns**
A collection of all variable definitions to the specific variable.

**Return type**
list

**find_consumers**(*var_def*, *simplified_graph=True*)

Find all consumers to the specified variable definition.

**Parameters**

- **var_def** (ProgramVariable) – The variable definition.
- **simplified_graph** (*bool*) – True if we want to search in the simplified graph, False otherwise.

**Returns**
A collection of all consumers to the specified variable definition.

**Return type**
list

**find_killers**(*var_def*, *simplified_graph=True*)

Find all killers to the specified variable definition.

**Parameters**

- **var_def** (ProgramVariable) – The variable definition.
- **simplified_graph** (*bool*) – True if we want to search in the simplified graph, False otherwise.

**Returns**
A collection of all killers to the specified variable definition.

**Return type**
list

**find_sources**(*var_def*, *simplified_graph=True*)

Find all sources to the specified variable definition.

**Parameters**

- **var_def** (ProgramVariable) – The variable definition.
- **simplified_graph** (*bool*) – True if we want to search in the simplified graph, False otherwise.

**Returns**
A collection of all sources to the specified variable definition.

**Return type**
list

### 10.3.19 angr.analyses.decompiler

**class** angr.analyses.decompiler.**AILSimplifier**

Bases: *Analysis*

Perform function-level simplifications.

**__init__**(*func*, *func_graph*, *ail_manager*, *remove_dead_memdefs=False*, *stackarg_offset_manager=None*, *unify_variables=False*, *gp=None*, *narrow_expressions=False*, *fold_expressions=True*, *only_consts=False*, *fold_callexprs_into_conditions=False*, *use_callee_saved_regs_at_return=True*, *rewrite_ccalls=True*, *rename_ccalls=True*, *rewrite_dirty=True*, *removed_vvar_ids=None*, *arg_vvars=None*, *avoid_vvar_ids=None*)

**class** angr.analyses.decompiler.**BaseStructuredCodeGenerator**

Bases: object

**__init__**(*flavor=None*, *notes=None*, *expr_comments=None*, *stmt_comments=None*, *const_formats=None*)

**static adjust_mapping_positions**(*offset*, *pos_to_node*, *pos_to_addr*, *addr_to_pos*)

Adjust positions in the mappings to account for the notes that are prepended to the text.

**Parameters**

- **offset** (int) – The length of the notes to prepend.
- **pos_to_node** (*PositionMapping*) – The position to node mapping.
- **pos_to_addr** (*PositionMapping*) – The position to address mapping.
- **addr_to_pos** (*InstructionMapping*) – The address to position mapping.

**Return type**
tuple[*PositionMapping*, *PositionMapping*, *InstructionMapping*]

**Returns**
Adjusted mappings.

**reapply_options**(*options*)

**regenerate_text**()

**Return type**
None

**reload_variable_types**()

**Return type**
None

**next_ident**(*key*)

**Return type**
str

**Parameters**
**key** (*str*)

**reset_ident_counters**()

**Return type**
None

**next_node_idx**()

**Return type**
int

**class** angr.analyses.decompiler.**BlockSimplifier**

Bases: object

Simplify an AIL block.

Deliberately not an Analysis: it is instantiated once per block, hundreds of times per decompilation, so it skips the analysis-factory ceremony. Instantiate it directly with the project as the first argument; exceptions always propagate.

**__init__**(*project*, *block*, *ail_manager*, *func_addr=None*, *stack_pointer_tracker=None*, *peephole_optimizations=None*, *preserve_vvar_ids=None*, *type_hints=None*, *cached_reaching_definitions=None*, *cached_propagator=None*, *peephole_bundle=None*)

**Parameters**

- **block** (Block | None) – The AIL block to simplify. Setting it to None to skip calling self._analyze(), which is useful in test cases.
- **peephole_bundle** (*PeepholeOptimizationBundle* | None) – A pre-built PeepholeOptimizationBundle to reuse. Its construction parameters must match this BlockSimplifier's; callers that simplify many blocks should build one bundle and pass it to every BlockSimplifier they create.
- **project** (Project)
- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **peephole_optimizations** (*Iterable[type[PeepholeOptimizationStmtBase | PeepholeOptimizationExprBase | PeepholeOptimizationMultiStmtBase]] | None*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)

**static replace_and_build**(*block*, *replacements*, *ail_manager*, *replace_assignment_dsts=False*, *replace_loads=False*, *gp=None*, *replace_registers=True*, *max_expr_depth=13*, *max_stmt_size=20000*)

**Return type**
tuple[bool, Block]

**Parameters**

- **block** (*Block*)
- **replacements** (*Mapping[AILCodeLocation, Mapping[Expression, Expression]]*)
- **ail_manager** (*Manager*)

- **replace_assignment_dsts** (*bool*)
- **replace_loads** (*bool*)
- **gp** (*int | None*)
- **replace_registers** (*bool*)
- **max_expr_depth** (*int | None*)
- **max_stmt_size** (*int*)

**class** angr.analyses.decompiler.**CStructuredCodeGenerator**

Bases: *BaseStructuredCodeGenerator*, *Analysis*, *Serializable*

**__init__**(*func*, *sequence*, *indent=0*, *cfg=None*, *func_args=None*, *binop_depth_cutoff=16*, *show_casts=True*, *braces_on_own_lines=True*, *use_compound_assignments=True*, *show_local_types=True*, *comment_gotos=False*, *cstyle_null_cmp=True*, *flavor=None*, *stmt_comments=None*, *expr_comments=None*, *show_externs=True*, *externs=None*, *const_formats=None*, *show_demangled_name=True*, *show_disambiguated_name=True*, *ail_graph=None*, *simplify_else_scope=True*, *cstyle_ifs=True*, *omit_func_header=False*, *display_block_addrs=False*, *display_vvar_ids=False*, *min_data_addr=4194304*, *notes=None*, *display_notes=True*, *max_str_len=None*, *prettify_thiscall=False*, *cstyle_void_param=True*, *indent_size=4*, *variable_map=None*, *stl_accessor_calls=False*)

**Parameters**

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)
- **min_data_addr** (*int*)
- **display_notes** (*bool*)
- **max_str_len** (*int | None*)
- **prettify_thiscall** (*bool*)
- **cstyle_void_param** (*bool*)
- **indent_size** (*int*)
- **variable_map** (*VariableMap | None*)
- **stl_accessor_calls** (*bool*)

**reapply_options**(*options*)

**cleanup**()

Remove existing rendering results.

**regenerate_text**()

Re-render text and re-generate all sorts of mapping information.

**Return type**
None

**render_text**(*cfunc*)

**Return type**
*RenderResult*

**Parameters**
**cfunc** (*CFunction*)

**render_notes**()

Render decompilation notes.

**Return type**
str

**Returns**
A string containing all notes.

**reload_variable_types**()

**Return type**
None

**default_simtype_from_bits**(*n*, *signed=True*)

**Return type**
*SimType*

**Parameters**

- **n** (*int*)
- **signed** (*bool*)

**variables_unify**(*v1*, *v2*)

**Return type**
bool

**Parameters**

- **v1** (VirtualVariable)
- **v2** (VirtualVariable)

**ERRNO_LOCATION_FUNCS = frozenset({'__errno', '__errno_location', '__error', '_errno'})**

The libc functions that return `&errno`. `errno` is a macro that dereferences one of them, so it never survives into a binary as a symbol; what reaches decompilation is `*(__errno_location())`, which is both correct and unreadable.

**class** angr.analyses.decompiler.**CallSiteMaker**

Bases: object

Add calling convention, declaration, and args to a call site.

**__init__**(*project*, *block*, ***, *ail_manager*, *reaching_definitions=None*, *stack_pointer_tracker=None*)

**Parameters**

- **project** (Project)
- **ail_manager** (*Manager*)
- **reaching_definitions** (SRDAModel | *None*)

**class** angr.analyses.decompiler.**Clinic**

Bases: *Analysis*, *Serializable*

A Clinic deals with AILments: it lifts a function to AIL and runs the decompiler's simplification pipeline on it.

AIL graphs exposed after a DECOMPILE-mode run:

- `cc_graph`: the graph at the end of Clinic's own simplification, frozen as a copy before region identification. Serialized; available on both live and cached-and-reloaded clinics.
- `graph`: `cc_graph` further transformed by the decompiler's graph-simplification passes, region identification, and region-simplification passes — the final graph. Serialized; available on both live and cached-and-reloaded clinics.
- `unoptimized_graph`: a copy taken before the first structure-altering optimization pass; use it for an exact instruction-to-AIL mapping. Only built (and serialized) with `Decompiler(save_unoptimized_graph=True)`; otherwise it is None on both live and cached-and-reloaded clinics.
- `_ail_graph` / `_init_ail_graph`: pipeline internals; never serialized.

**__init__**(*func*, *remove_dead_memdefs=False*, *exception_edges=False*, *sp_tracker_track_memory=True*, *fold_expressions=True*, *fold_callexprs_into_conditions=False*, *insert_labels=True*, *optimization_passes=None*, *cfg=None*, *peephole_optimizations=None*, *must_struct=None*, *reset_variable_names=False*, *rewrite_ites_to_diamonds=True*, *rewrite_ites_to_diamond_max_cases=15*, *cache=None*, *mode=ClinicMode.DECOMPILE*, *sp_shift=0*, *inline_functions=None*, *inlined_counts=None*, *inlining_parents=None*, *vvar_id_start=0*, *optimization_scratch=None*, *desired_variables=None*, *force_loop_single_exit=True*, *refine_loops_with_single_successor=False*, *expose_loop_head_backedges=False*, *typehoon_cls=<class 'angr.analyses.typehoon.typehoon.Typehoon'>*, *max_ail_statements=1000000*, *max_type_constraints=100000*, *type_constraint_set_degradation_threshold=150*, *ail_graph=None*, *arg_vvars=None*, *start_stage=ClinicStage.INITIALIZATION*, *end_stage=None*, *skip_stages=()*, *notes=None*, *static_vvars=None*, *static_buffers=None*, *flatten_args=False*, *constrain_callee_prototypes=False*, *semvar_naming=True*, *flavor='pseudocode'*, *variable_map=None*, *save_unoptimized_graph=False*, *known_patterns=None*, *recognize_known_patterns=True*)

**Parameters**

- **func** (Function)
- **peephole_optimizations** (*Iterable[type[PeepholeOptimizationStmtBase | PeepholeOptimizationExprBase]] | None*)
- **must_struct** (*set[str] | None*)
- **rewrite_ites_to_diamond_max_cases** (*int*)
- **cache** (DecompilationCache | *None*)
- **mode** (ClinicMode)
- **sp_shift** (*int*)
- **inline_functions** (*set[Function] | None*)
- **inlined_counts** (*dict[int, int] | None*)
- **inlining_parents** (*set[int] | None*)
- **vvar_id_start** (*int*)
- **optimization_scratch** (*dict[str, Any] | None*)
- **desired_variables** (*set[str] | None*)
- **force_loop_single_exit** (*bool*)
- **refine_loops_with_single_successor** (*bool*)

- **expose_loop_head_backedges** (*bool*)
- **max_ail_statements** (*int*)
- **max_type_constraints** (*int*)
- **type_constraint_set_degradation_threshold** (*int*)
- **ail_graph** (*DiGraph | None*)
- **arg_vvars** (*dict[int, tuple[VirtualVariable, SimVariable]] | None*)
- **start_stage** (*ClinicStage | None*)
- **end_stage** (*ClinicStage | None*)
- **skip_stages** (*tuple[ClinicStage, ...]*)
- **notes** (*dict[str, DecompilationNote] | None*)
- **static_vvars** (*dict | None*)
- **static_buffers** (*dict | None*)
- **constrain_callee_prototypes** (*bool*)
- **semvar_naming** (*bool*)
- **flavor** (*str*)
- **variable_map** (*VariableMap | None*)
- **save_unoptimized_graph** (*bool*)
- **known_patterns** (*str | tuple[str, ...] | None*)
- **recognize_known_patterns** (*bool*)

**block**(*addr*, *size*)

Get the converted block at the given specific address with the given size.

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**Returns**

**dbg_repr**()

**Returns**

**calculate_stack_depth**()

**copy_graph**(*graph=None*)

**Return type**
DiGraph

**parse_variable_addr**(*addr*)

**Return type**
tuple[Any, Any]

**Parameters**
**addr** (Expression)

**new_block_addr**()

Return a block address that does not conflict with any existing blocks.

**Return type**
int

**Returns**
The block address.

**static remove_empty_nodes**(*graph*)

**Return type**
DiGraph

**Parameters**
**graph** (*DiGraph*)

**constrain_callee_prototypes**()

Constrain the types of callee function arguments based on facts that are observed at call sites. Note that this function will change the prototypes of (callee) functions in the knowledge base, which means it may affect the decompilation output of the current function if it is decompiled again.

**resolve_peephole_optimizations**()

Retry resolving peephole-optimization names that were unresolvable at parse time (their defining module may have been imported since). Resolved classes move into peephole_optimizations; names that still do not resolve stay in unresolvable_peephole_optimizations.

**Return type**
None

**classmethod parse_from_cmessage**(*cmsg*, *, *project=None*, *kb=None*, *function=None*, *cfg=None*, ***kwargs*)

Bypasses Clinic.__init__() (which runs the analysis) and reconstructs the instance directly. Runtime back-references — project / kb / function / _cfg — come from kwargs.

Only the state consumed by the decompiler's cache-reuse path and by post-decompilation consumers (cc_graph, graph, optionally unoptimized_graph) is serialized; regenerable and runtime-only state is restored to its default here. If function is None and kb is provided, the function is resolved by address from the cmessage.

**class** angr.analyses.decompiler.**ClinicMode**

Bases: Enum

Analysis mode for Clinic.

**DECOMPILE = 1**

**COLLECT_DATA_REFS = 2**

**class** angr.analyses.decompiler.**DecompilationCache**

Bases: *Serializable*

Caches key data structures that can be used later for refining decompilation results, such as retyping variables.

**__init__**(*addr*)

**parameters: dict[str, Any]**

**version: str**

**timestamp:** *int*

**addr**

**cfg:** *CFGModel* | None

**type_constraints:** dict[*TypeVariable*, set[*TypeConstraint*]]

**arg_vvars:** dict

**func_typevar:** *TypeVariable* | None

**var_to_typevar:** dict

**stackvar_max_sizes:** dict

**stack_offset_typevars:** dict

**codegen:** *BaseStructuredCodeGenerator* | None

**clinic:** *Clinic* | None

**variable_map:** *VariableMap* | None

**ite_exprs:** set[tuple[int, *Expression*]]

**binop_operators:** dict[*OpDescriptor*, str]

**errors:** list[str]

**function_summary:** str | None

**notes:** dict[str, *DecompilationNote*]

**max_tv_id:** int

**property local_types**

**classmethod parse_from_cmessage**(*cmsg*, *, *project=None*, *kb=None*, *function=None*, *cfg=None*, **_)

Parse a DecompilationCache from a cmessage. Runtime back-references (project, kb, function, cfg) are passed through to the embedded Clinic / codegen parsers so the parsed cache is functional for cache-hit validity checks. Decompilation variables live on kb.dec_variables.

**class** angr.analyses.decompiler.**Decompiler**

Bases: *Analysis*

The decompiler analysis.

Run this on a Function object for which a normalized CFG has been constructed. The fully processed output can be found in result.codegen.text

AIL graphs exposed on the result (both on a fresh run and on a cache hit, including caches reloaded from angrdb or the runtime-db spill):

- `ail_graph` (= `clinic.cc_graph`): the simplified graph before region identification.
- `clinic.graph`: the final graph after region identification and region simplification.
- `unoptimized_ail_graph` (= `clinic.unoptimized_graph`): a snapshot before the first structure-altering optimization pass; use it for an exact instruction-to-AIL mapping. Only built when `save_unoptimized_graph=True` is passed; otherwise this attribute is None on both fresh runs and cache hits.

Complexity guard: pathologically large functions (e.g. packer filler that linear disassembly turns into thousands of blocks) can keep the pipeline busy indefinitely, so two size limits are checked before the expensive work starts. `max_function_blocks` (default 50000, `DEFAULT_MAX_FUNCTION_BLOCKS`) bounds the number of CFG basic blocks and is checked before any AIL is built; `max_ail_statements` (default 1000000, `DEFAULT_MAX_AIL_STATEMENTS`) bounds the number of AIL statements and is checked right after the AIL graph is built. Both are ~10x the largest function in angr's test binary corpus. Either limit can be changed through the decompilation option of the same name, and setting one to 0 disables it. When a limit is exceeded, decompilation stops with an `AngrDecompilationComplexityError` naming the limit and the actual size: with `fail_fast=True` it is raised, otherwise it is recorded in `self.errors` (and in `kb.decompilations[...].errors`) and exposed as `complexity_error`, and `codegen` stays None.

The optimization passes come from `preset` (a name or a DecompilationPreset), or from an explicit `optimization_passes` list when no preset is given. `disable_opts` drops individual passes from the preset's list, which is how a caller asks for "this preset, minus that one pass" without restating it.

**__init__**(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *disable_opts=None*, *sp_tracker_track_memory=True*, *peephole_optimizations=None*, *vars_must_struct=None*, *flavor='pseudocode'*, *expr_comments=None*, *stmt_comments=None*, *ite_exprs=None*, *binop_operators=None*, *decompile=True*, *regen_clinic=False*, *inline_functions=None*, *desired_variables=None*, *update_memory_data=True*, *want_full_graph=False*, *generate_code=True*, *use_cache=True*, *update_cache=True*, *expr_collapse_depth=16*, *clinic_graph=None*, *clinic_arg_vvars=None*, *clinic_start_stage=None*, *clinic_end_stage=None*, *clinic_skip_stages=()*, *static_vvars=None*, *static_buffers=None*, *codegen_cls=<class 'angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator'>*, *save_unoptimized_graph=False*)

**Parameters**

- **func** (Function | *str* | *int*)
- **cfg** (CFGFast | CFGModel | *None*)
- **preset** (*str* | DecompilationPreset | *None*)
- **peephole_optimizations** (*Iterable*[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase]] | *None*)
- **vars_must_struct** (*set*[*str*] | *None*)
- **update_memory_data** (*bool*)
- **want_full_graph** (*bool*)
- **generate_code** (*bool*)
- **use_cache** (*bool*)
- **update_cache** (*bool*)
- **expr_collapse_depth** (*int*)
- **static_vvars** (*dict* | *None*)
- **static_buffers** (*dict* | *None*)
- **save_unoptimized_graph** (*bool*)

**property complexity_error:** *AngrDecompilationComplexityError* | None

The complexity-limit violation that aborted this decompilation, or None if no limit was exceeded. Inspect `limit_name`, `limit`, and `actual` on it to decide whether to raise the limit or skip the function.

**reflow_variable_types**(*cache*)

Re-run type inference on an existing variable recovery result, then rerun codegen to generate new results.

**Returns**

**Parameters**
**cache** (DecompilationCache)

**find_data_references_and_update_memory_data**(*seq_node*)

**Parameters**
**seq_node** (SequenceNode)

**transform_graph_from_ssa**(*ail_graph*)

Translate an SSA AIL graph out of SSA form. This is useful for producing a non-SSA AIL graph for displaying in angr management.

**Parameters**
**ail_graph** (DiGraph) – The AIL graph to transform out of SSA form.

**Return type**
DiGraph

**Returns**
The translated AIL graph.

**transform_seqnode_from_ssa**(*seq_node*)

**Return type**
*SequenceNode*

**Parameters**
**seq_node** (SequenceNode)

**llm_refine**()

Use the configured LLM to suggest improved variable names, function names, and variable types. Returns True if any changes were made.

**Return type**
bool

**llm_suggest_variable_names**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest better variable names for the decompiled code. Returns True if any variables were renamed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller. If False (default), exceptions are caught and the method returns False.
- **code_text** (*str | None*)

**Return type**
bool

**llm_suggest_function_name**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest a better function name. Only suggests rename for auto-generated names (starting with sub_ or fcn.). Returns True if the function was renamed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.

- **code_text** (*str | None*)

**Return type**
bool

**llm_suggest_variable_types**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest better C types for variables. Returns True if any variable types were changed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
bool

**llm_summarize_function**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to produce a natural-language summary of what the decompiled function does. The summary is stored in the DecompilationCache and returned.

Returns the summary string, or None if summarization failed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
str | None

**static options_to_params**(*options*)

Convert decompilation options to a dict of params.

**Parameters**
**options** (list[tuple[*DecompilationOption*, Any]]) – The decompilation options.

**Return type**
dict[str, Any]

**Returns**
A dict of keyword arguments.

**class** angr.analyses.decompiler.**GraphDephication**

Bases: *DephicationBase*

GraphDephication removes phi expressions from an AIL graph, essentially transforms a partial-SSA form of AIL graph to a normal AIL graph.

**__init__**(*func*, *ail_graph*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_map=None*)

**Parameters**

- **func** (*Function* | str) – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_map** (VariableMap | *None*)

class angr.analyses.decompiler.**ImportSourceCode**

Bases: *BaseStructuredCodeGenerator*, *Analysis*

**__init__**(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

**regenerate_text**()

class angr.analyses.decompiler.**KnownPattern**

Bases: object

A known code idiom, described declaratively.

**Variables**

- **name** – Identifier, e.g. "std_string_length".
- **display_name** – Human-readable name, e.g. "std::string::length".
- **call_name** – Target name of the synthesized Call.
- **pattern** – The structural pattern to match.
- **params** – Ordered parameters; their captures become the call arguments, in order.
- **returnty** – Return type of the synthesized call.
- **returnty_factory** – Optional callable building a call-site-specific return type from the values of the constant extra_args captures (e.g. CONTAINING_RECORD returns a pointer to a record whose layout depends on the matched field offset). Falls back to returnty when it returns None or when the constant values are unavailable.
- **extra_args** – Names of (typically constant) captures appended to the call arguments after params.
- **arches** – Allowed archinfo.Arch.name values; None = any.
- **platforms** – Allowed OS names (project.simos.name, lowercased); None = any.
- **where** – Optional cross-capture predicate evaluated on the bindings after a structural match.
- **collapse_capture** – Optional capture name identifying the idiom that a match belongs to: within one function, matches binding it to the same virtual variable all describe *one* source-level idiom, and only the one with the largest collapse_max_capture constant is kept. The compiler folds CONTAINING_RECORD's subtraction into every field displacement, so one source idiom otherwise emits a call per field, each naming a different (and mostly wrong) record base.
- **collapse_max_capture** – Capture whose constant value orders the collapsing; required when collapse_capture is set.
- **binary_guard** – Optional predicate on the Project; the pattern is only used when it returns True (e.g. require C++ evidence for STL patterns to avoid false positives on plain C binaries).
- **default_enabled** – Whether the pattern is used when the caller does not explicitly select patterns. Generic patterns prone to false positives should set this False.

**name:** *str*

**display_name:** *str*

**call_name:** *str*

**pattern:** *PatternExpr* | *PatternStmt* | *PGraphPat*

**params:** tuple[*PatternParam*, ...]

**returnty:** str | *CppRef* | None = None

**returnty_factory:** Callable[[Arch, dict[str, int]], *SimType* | None] | None = None

**extra_args:** tuple[str, ...] = ()

**arches:** tuple[str, ...] | None = None

**platforms:** tuple[str, ...] | None = None

**where:** Callable[[dict[str, *Expression*]], bool] | None = None

**collapse_capture:** str | None = None

**collapse_max_capture:** str | None = None

**binary_guard:** Callable[[*Project*], bool] | None = None

**default_enabled:** bool = True

**suppressed_by:** tuple[str, str, str] | None = None

Drop a match of this pattern when a pattern whose name matches the regex matched on the same object: `(name regex, my capture, their capture)`. The two captures must bind `.likes()`-equal expressions. This is how a generic accessor yields to a specific one on the same base: a bare `Load(s + 8)` is std::string::length on a string and _M_finish on a vector, and the vector's other fields say which. Judged against every match of the suppressing family, selected or not, so it cannot interact with largest-match selection.

**claims_only:** bool = False

Match only to claim a base for `suppressed_by` purposes; never outlined, never rewritten, never reported. For shapes that identify an object beyond doubt but are not worth naming: a vector's capacity check says "this is a vector" and nothing a reader wants a call for.

**pure_calls:** frozenset[str] = frozenset({})

Callees this pattern reads through *PCallResult* whose only effect is their result: once the pattern has consumed that result and nothing else does, the call goes too. Naming a callee in a pattern is already a claim about what it computes; this is the claim that it computes nothing else. `__ctype_b_loc` returns a table pointer, and a bare `__ctype_b_loc();` left behind after `isspace(c)` is noise.

**applicable**(*arch_name*, *platform*)

**Return type**
bool

**Parameters**

- **arch_name** (*str*)
- **platform** (*str | None*)

**const_args_of_call**(*call*)

Recover the `extra_args` constant values from a synthesized call's trailing arguments. Returns None when the call does not carry them.

**Return type**
dict[str, int] | None

**Parameters**
**call** (Call)

**prototype**(*arch*, *const_args=None*)

Build the synthesized call's prototype. `const_args` (the values of the constant `extra_args` captures, see `const_args_of_call()`) enables call-site-specific return types via `returnty_factory`. Returns None if any declared type fails to resolve.

**Return type**
*SimTypeFunction* | None

**Parameters**

- **arch** (*Arch*)
- **const_args** (*dict[str, int] | None*)

**__init__**(*name*, *display_name*, *call_name*, *pattern*, *params*, *returnty=None*, *returnty_factory=None*, *extra_args=()*, *arches=None*, *platforms=None*, *where=None*, *collapse_capture=None*, *collapse_max_capture=None*, *binary_guard=None*, *default_enabled=True*, *suppressed_by=None*, *claims_only=False*, *pure_calls=frozenset({})*)

**Parameters**

- **name** (*str*)
- **display_name** (*str*)
- **call_name** (*str*)
- **pattern** (PatternExpr | PatternStmt | PGraphPat)
- **params** (*tuple[PatternParam, ...]*)
- **returnty** (*str | CppRef | None*)
- **returnty_factory** (*Callable[[Arch, dict[str, int]], SimType | None] | None*)
- **extra_args** (*tuple[str, ...]*)
- **arches** (*tuple[str, ...] | None*)
- **platforms** (*tuple[str, ...] | None*)
- **where** (*Callable[[dict[str, Expression]], bool] | None*)
- **collapse_capture** (*str | None*)
- **collapse_max_capture** (*str | None*)
- **binary_guard** (*Callable[[Project], bool] | None*)
- **default_enabled** (*bool*)
- **suppressed_by** (*tuple[str, str, str] | None*)
- **claims_only** (*bool*)
- **pure_calls** (*frozenset[str]*)

**Return type**
None

**class** angr.analyses.decompiler.**KnownPatternFinder**

Bases: *Analysis*

Finds occurrences of KnownPatterns in a Clinic AIL graph.

The graph to match on is the final (SSA-form) Clinic graph, i.e. `decompiler.ail_graph` / `clinic.cc_graph`. Matches are reported in `self.matches`; `outline()` turns a match into a synthesized call by invoking the Outliner analysis on a copy of the graph.

**MAX_REMOTE_CHASE_DEPTH = 6**

**__init__**(*func*, *ail_graph*, *patterns=None*, *chase_defs=True*, *chase_remote_defs=True*, *skip_conversions=True*, *vvar_id_start=48879*, *block_addr_start=2864381952*, *ail_manager=None*, *force_patterns=None*)

`patterns` may be pattern templates (KnownPatternTemplate) and/or already-concrete KnownPatterns; templates are instantiated for this binary's PatternContext. When None, the registry is used, filtered down to the templates that are enabled for this target: default-on ones, ones named by `force_patterns`, and ones whose gate opens (see *gating*).

`force_patterns` is the user's force-enable selection, `"all"` or an iterable of template names / call names, and is only consulted when `patterns` is None (an explicit pattern list is already a selection).

**Parameters**

- **func** (Function)
- **ail_graph** (*DiGraph*)
- **patterns** (*Iterable[object] | None*)
- **chase_defs** (*bool*)
- **chase_remote_defs** (*bool*)
- **skip_conversions** (*bool*)
- **vvar_id_start** (*int*)
- **block_addr_start** (*int*)
- **force_patterns** (*str | Iterable[str] | None*)

**call_target_names**(*call*)

Public face of `_resolve_call_target()`, for the outliner's clean-up of calls a pattern declared pure.

**Return type**
frozenset[str]

**Parameters**
**call** (Call)

**apply_matches**(*matches*, *ail_graph*)

Apply as many of `matches` as possible, batching where it matters.

Applying one match at a time is fine until several of them interleave in one block, which is exactly what consecutive field swaps do:

**t1 = x->b; t2 = y->b; y->b = t1; x->b = t2;**
^ the next field's load sits in this one's gap

Rewrite the first and the second's statements have moved; rewrite it anyway and the third now has an opaque call sitting in its gap, which no disjointness test will let it move across. So the void statement-span matches of a block, which the finder has already checked do not conflict, are spliced in together, against the statement indices they were found at. Everything else keeps the one-at-a-time path.

Returns the new graph and the ids of the matches that were applied.

**Return type**
tuple[DiGraph, set[int]]

**Parameters**

- **matches** (*list[KnownPatternMatch]*)
- **ail_graph** (*DiGraph*)

**outline**(*match*, *ail_graph=None*)

Outline one match: split the anchor block so the matched span forms its own block, invoke the Outliner analysis on it, and rewrite the synthesized callsite into the pattern's call. Operates on (and returns) a copy; neither `self._graph` nor `ail_graph` is mutated.

**Return type**
*OutlineResult*

**Parameters**

- **match** (KnownPatternMatch)
- **ail_graph** (*DiGraph | None*)

**MAX_LIFTED_EXPR_NODES = 8**

Node budget for a lifted parameter expression; see _is_liftable.

**outline_at**(*location*, *pattern*, *ail_graph*)

Find `pattern` at `location` (`(block_addr, block_idx)` or `(block_addr, block_idx, stmt_idx)`) in `ail_graph` and outline it.

**Return type**
*OutlineResult*

**Parameters**

- **location** (*tuple[int, int | None] | tuple[int, int | None, int]*)
- **pattern** (KnownPattern)
- **ail_graph** (*DiGraph*)

**class** angr.analyses.decompiler.**KnownPatternMatch**

Bases: object

One occurrence of a KnownPattern in an AIL graph.

For expression-level matches, the boundary is `consumed_stmt_idxs` (statements wholly owned by the match, e.g. chased definitions) plus `expr_path` (the location of the matched sub-expression inside the anchor statement). For statement-sequence matches, `stmt_span` holds the ordered matched statement indices (the anchor is the first) and `matched_expr` is None.

`duplicated_stmt_idxs` holds chased definitions that are copied into the outlined region instead of moved: the originals stay where they are (other code still uses them) and the copies make the pattern's captured values the region's live-ins. It is always disjoint from `consumed_stmt_idxs`.

`recomputed_defs` does the same for definitions that live in *another* block (a CSE'd value, possibly reaching the anchor through a phi): the expression is pure, so the region recomputes it from the captured operands.

**pattern:** *KnownPattern*

**block_loc:** `tuple[int, int | None]`

**anchor_stmt_idx:** `int`

**expr_path:** `tuple[tuple[str, int | None], ...]`

**consumed_stmt_idxs:** frozenset[int]

**captures:** dict[str, *Expression*]

**matched_expr:** *Expression* | None

**duplicated_stmt_idxs:** frozenset[int] = frozenset({})

**recomputed_defs:** tuple[tuple[int, *Expression*], ...] = ()

**base_aliases:** tuple[tuple[int, int, int], ...] = ()

**stmt_span:** tuple[int, ...] | None = None

**block_map:** dict[str, tuple[int, int | None]] | None = None

**consumed_by_block:** dict[tuple[int, int | None], frozenset[int]] | None = None

**frontier_locs:** frozenset[tuple[int, int | None]] | None = None

**__init__**(*pattern*, *block_loc*, *anchor_stmt_idx*, *expr_path*, *consumed_stmt_idxs*, *captures*, *matched_expr*, *duplicated_stmt_idxs=frozenset({})*, *recomputed_defs=()*, *base_aliases=()*, *stmt_span=None*, *block_map=None*, *consumed_by_block=None*, *frontier_locs=None*)

**Parameters**

- **pattern** (KnownPattern)
- **block_loc** (*tuple[int, int | None]*)
- **anchor_stmt_idx** (*int*)
- **expr_path** (*tuple[tuple[str, int | None], ...]*)
- **consumed_stmt_idxs** (*frozenset[int]*)
- **captures** (*dict[str, Expression]*)
- **matched_expr** (Expression | *None*)
- **duplicated_stmt_idxs** (*frozenset[int]*)
- **recomputed_defs** (*tuple[tuple[int, Expression], ...]*)
- **base_aliases** (*tuple[tuple[int, int, int], ...]*)
- **stmt_span** (*tuple[int, ...] | None*)
- **block_map** (*dict[str, tuple[int, int | None]] | None*)
- **consumed_by_block** (*dict[tuple[int, int | None], frozenset[int]] | None*)
- **frontier_locs** (*frozenset[tuple[int, int | None]] | None*)

**Return type**
None

**class** angr.analyses.decompiler.**RegionIdentifier**

Bases: *Analysis*

A region is a single-entry-single-exit subgraph of control flow. The region identifier recursively identifies the smallest possible regions within a function graph and creates a RegionOverlay object whose nodes are either Blocks or RegionOverlays.

Note, that the analysis may modify the graph in-place. If you want to keep the original graph, set the *update_graph* parameter to False.

**__init__**(*func*, *cond_proc=None*, *graph=None*, *ail_manager=None*, *update_graph=True*, *largest_successor_tree_outside_loop=True*, *force_loop_single_exit=True*, *refine_loops_with_single_successor=False*, *expose_loop_head_backedges=False*, *entry_node_addr=None*)

**test_reducibility**()

**Return type**
bool

**class** angr.analyses.decompiler.**RegionOverlay**

Bases: Generic

A single-entry region marked over the complete graph held by an OverlayManager. The region tree is built out of RegionOverlay objects (RegionIdentifier emits them) and they are the only region type the decompiler uses.

Overlays form a tree (nested, never overlapping). An overlay's members are either complete-graph nodes it owns directly or child overlays. The region graph and the region graph-with-successors are derived on demand from the complete graph by quotienting child overlays into single nodes; *successors* are likewise derived from edges that cross the region boundary, so they can never go stale.

Read-only API (`head`, `graph`, `graph_with_successors`, `successors`, `cyclic`, `addr`) mirrors the region container consumers expect; `graph`/`graph_with_successors`/`successors` are lazy zero-copy views.

Mutation verbs:

- true structural changes (`add_node`, `remove_node`, `add_edge`, `detach_edge`, `replace_nodes`) pass through to the complete graph, so the effects are immediately visible to all enclosing regions;
- `hide_edge` removes an edge from this overlay's views only (the old "remove it from region graphs but keep the parent edge" pattern);
- `finalize(result_node)` collapses a fully-structured region into a single node of its parent;
- `dissolve()` merges an unsuccessfully-structured region back into its parent.

**__init__**(*mgr*, *head*, *cyclic*, *cyclic_ancestor=False*, *parent=None*)

**Parameters**

- **mgr** (*OverlayManager[T]*)
- **head** (*Tx[T] | None*)
- **cyclic** (*bool*)
- **cyclic_ancestor** (*bool*)
- **parent** (*RegionOverlay | None*)

**head**

**cyclic**

**cyclic_ancestor**

**parent**

**children:** list[*RegionOverlay*]

**edge_marks:** defaultdict[str, set[tuple[*Tx*, *Tx*]]]

**replacement**

**property addr**

**property manager:** *OverlayManager*[T]

**property members:** set[*Tx*[T]]

**ancestors**()

All overlays enclosing this one, including this one itself.

**Return type**
set[*RegionOverlay*]

**underlying_nodes**()

All complete-graph nodes inside this region (including nodes of nested regions).

**Return type**
set[TypeVar(T, bound= *RegionBound*)]

**create_subregion**(*head*, *members*, *cyclic*, *cyclic_ancestor=False*)

Carve a new child overlay out of this overlay. `members` must be a subset of this overlay's members (complete-graph nodes owned by this overlay and/or existing child overlays); `head` must be one of them.

**Return type**
*RegionOverlay*[TypeVar(T, bound= *RegionBound*)]

**Parameters**

- **head** (*Tx[T]*)
- **members** (*Iterable*)
- **cyclic** (*bool*)
- **cyclic_ancestor** (*bool*)

**representative**(*node*)

The node standing for a complete-graph node in this overlay's with-successors view.

**Return type**
*Tx*[TypeVar(T, bound= *RegionBound*)] | None

**Parameters**
**node** (*T*)

**successor_nodes**()

The derived successor set of this region: representatives of all complete-graph nodes targeted by edges leaving the region.

**Return type**
set[*Tx*[TypeVar(T, bound= *RegionBound*)]]

**view_graph**(*full=False*, *include_marked=False*, *blacklisted_edges=frozenset({})*)

A zero-copy, networkx-compatible view of this region (see RegionOverlayGraph).

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**Parameters**

- **full** (*bool*)
- **include_marked** (*bool*)
- **blacklisted_edges** (*frozenset*)

**view**()

The region graph (members only): a zero-copy networkx-compatible view; treat it as read-only.

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**view_with_successors**()

The region graph including successor nodes: a zero-copy view; treat it as read-only.

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**property raw_graph:** *RegionOverlayGraph*[T]

The member view including marked edges (the old graph with cyclic_refinement_outgoing attrs present).

**property raw_graph_with_successors:** *RegionOverlayGraph*[T]

The with-successors view including marked edges.

**property graph:** networkx.DiGraph[*Tx*[T]]

**property graph_with_successors:** networkx.DiGraph[*Tx*[T]]

**property successors:** set[*Tx*[T]]

**property full_graph:** None

**complete_graph_entry_count**(*node*)

The number of edges entering a node in the complete graph, counted over every region. For a region object, the entries of its head from outside the region (its own back edges do not count).

**Return type**
int

**Parameters**
**node** (Tx[*T*])

**external_entry_count**(*node*)

The number of edges entering a node (a successor of this region) from nodes that are not under this region: the entries it keeps once every edge from this region to it has been virtualized.

**Return type**
int

**Parameters**
**node** (Tx[*T*])

**add_node**(*node*)

Insert a new node into the complete graph as a direct member of this region.

**Return type**
None

**remove_node**(*node*, *absorbed_into=None*, *absorb_out_edges=True*)

Remove a node. If `node` is a member (or a member overlay's node), it is removed from the complete graph for real. If it is a successor of this region, the removal is interpreted as hiding all edges from this region to it (the successor belongs to an enclosing region and must survive).

When the node's content has been absorbed into another node during structuring, pass that node as `absorbed_into`: in-edges from outside this region (e.g. abnormal loop entries) are then rewired to it instead of being dropped, so enclosing regions keep their entry edges. With `absorb_out_edges` (the default), out-edges crossing the region boundary (e.g. loop exits) are rewired to it as well, so the complete graph never loses the region's exit flow; pass False only when the caller re-establishes every successor edge explicitly.

**Return type**
None

**Parameters**

- **node** (*Tx[T]*)
- **absorbed_into** (*Tx[T] | None*)
- **absorb_out_edges** (*bool*)

**hide_edge_to_successor**(*succ*)

**Return type**
None

**underlying_edge_pairs**(*src*, *dst*)

**Return type**
list[tuple[*Tx*[TypeVar(T, bound= *RegionBound*)], *Tx*[TypeVar(T, bound= *RegionBound*)]]]

**Parameters**

- **src** (*Tx[T]*)
- **dst** (*Tx[T]*)

**add_edge**(*src*, *dst*, ***data*)

Add a real edge to the complete graph. Overlay endpoints are resolved to underlying nodes: the destination resolves to its entry (head chain); overlay sources are not supported. Endpoints that are not in the complete graph yet become members of this region (mirroring networkx's implicit node creation).

**Return type**
None

**Parameters**

- **src** (*Tx[T]*)
- **dst** (*Tx[T]*)

**detach_raw_edge**(*u*, *v*)

Remove exactly one complete-graph edge. Unlike detach_edge, other edges between the view nodes representing u and v are kept (e.g. a block that jumps both to a loop's entry and into its body).

**Return type**
None

**Parameters**

- **u** (*T*)

- **v** (*T*)

**detach_edge**(*src*, *dst*)

Remove an edge from the complete graph for real (e.g., when the edge has been virtualized into a goto). Overlay endpoints remove all underlying edges between the two node sets.

**Return type**
None

**Parameters**

- **src** (Tx[T])
- **dst** (Tx[T])

**mark_edge**(*src*, *dst*, ***attrs*)

Mark a view-level edge (e.g. cyclic_refinement_outgoing) so RegionOverlayGraph hides it by default. Marks live in overlay state, never reach the complete graph, and are remapped/cleared with the region.

**Return type**
None

**Parameters**

- **src** (Tx[T])
- **dst** (Tx[T])

**absorb_successor_into**(*succ*, *new_node*)

Absorb a successor node into a structured member node in this region's with-successors view only (the successor still belongs to an enclosing region): the successor's view out-edges are re-attached to the member node as view-only extra edges, then the successor disappears from this region's views.

**Return type**
None

**Parameters**

- **succ** (Tx[T])
- **new_node** (Tx[T])

**drop_edge_marks_from**(*node*, *key*)

Clear marks on all out-edges of a node (the new_node after a replace), undoably.

**Return type**
None

**Parameters**

- **node** (Tx[T])
- **key** (*str*)

**remove_edge_with_successors_only**(*src*, *dst*)

Hide an edge from the with-successors view only, leaving the member view and the complete graph alone (a rare asymmetric bookkeeping pattern in Phoenix's switch-case structuring).

**Return type**
None

**Parameters**

- **src** (Tx[T])

- **dst** (Tx[T])

**hide_edge**(*src*, *dst*)

Remove an edge from this overlay's views only. Enclosing regions still see the underlying edge(s).

**Return type**
None

**Parameters**

- **src** (Tx[T])
- **dst** (Tx[T])

**replace_nodes**(*old_node_0*, *new_node*, *old_node_1=None*, *self_loop=True*)

Replace one or two member nodes with a new node, preserving and rewiring all underlying edges (including edges from/to nodes outside this region, which is how results become visible to enclosing regions).

Mirrors StructurerBase.replace_nodes, with one deliberate difference: in-edges of `old_node_1` from outside this region are rewired to `new_node` instead of being dropped (they are entry edges owned by enclosing regions, e.g. abnormal loop entries).

**Return type**
None

**Parameters**

- **old_node_0** (Tx[T])
- **new_node** (Tx[T])
- **old_node_1** (Tx[T] | None)
- **self_loop** (bool)

**snapshot_successors**()

Capture this region's structural successors and how many member edges reach each, taken before the region is structured. finalize() uses it to re-establish the region-to-successor edges that structuring removes when it refines the corresponding control-flow edges into breaks, and to tell successors whose every member edge was virtualized into a goto (see detach_edge()).

**Return type**
dict[Tx[TypeVar(T, bound= RegionBound)], int]

**finalize**(*result_node=None*, *succ_snapshot=None*)

Collapse this fully-structured region into its parent: the region must consist of a single member node (the structuring result), which takes the region's place among the parent's members. Returns that node.

`succ_snapshot` (from snapshot_successors() before structuring) is used to re-establish the edges from result_node to the region's successors: structuring may have removed the live control-flow edges (refining them into breaks), but the structured region still flows to those successors and enclosing regions must see that.

A successor whose every member edge was virtualized into a goto (detach_edge()) is a pure goto target and must NOT be reconnected. Consider a loop with two exits, where `err` is shared with other loops of the function:

**while (1) {**
if (s->bsLive >= 8) break; // exit A: the loop successor if (s->strm->avail_in == 0) goto err;
// exit B: virtualized into a goto by cyclic refinement refill();

} uc = take_bits(); // A …

**err:**
retVal = BZ_DATA_ERROR; goto save_state_and_return;

The loop region has successors A and err. Cyclic refinement keeps A as the loop successor (a break) and turns the exit to err into a goto statement, removing that edge from the complete graph. Reconnecting err here would give the loop node two successors in the parent region, one real and one fictitious: the ITE schema finds no conditions on them, the sequence schema needs a single successor, and last-resort refinement keeps both edges because each target would be orphaned, so the parent region can never be structured. With only A reconnected, the loop node is sequenced with A as usual and err stays reachable through its gotos. Cyclic refinement never virtualizes an exit that is its target's only entry in the complete graph: the region bails and dissolves into its parent, where the target is a member and the acyclic schemas structure it with the loop.

**Parameters**

- **result_node** (*Tx[T] | None*)
- **succ_snapshot** (*set[Tx[T]] | None*)

**collapse_to**(*result_node*)

Collapse this region into its parent by replacing all of its member nodes with a single external result node (the structuring result). Used by structurers that compute their result without destructively reducing the complete graph (e.g. DreamStructurer): the region's members are still present, so their crossing in/out edges are rewired onto `result_node` and the members are removed. Returns result_node.

This is the non-self-collapsing counterpart of finalize(); the legacy GraphRegion path called replace_region() for the same purpose.

**Parameters**
**result_node** (*RegionOverlay[T]*)

**dissolve**()

Merge this region back into its parent (the failure path of structuring): members are reparented, and edges hidden in this region stay hidden in the parent (matching how partially-structured region graphs were merged back before).

**Return type**
None

**class** angr.analyses.decompiler.**RegionSimplifier**

Bases: *Analysis*

Simplifies a given region.

**__init__**(*func*, *region*, *ail_manager*, *arg_vvars=None*, *simplify_switches=True*, *simplify_ifelse=True*, *variable_manager=None*, *loopctr_naming=True*)

**Parameters**

- **ail_manager** (*Manager*)
- **arg_vvars** (*set[int] | None*)
- **simplify_switches** (*bool*)
- **simplify_ifelse** (*bool*)
- **variable_manager** (*VariableManagerInternal | None*)
- **loopctr_naming** (*bool*)

**class** angr.analyses.decompiler.**SeqNodeDephication**

Bases: *DephicationBase*

SeqNodeDephication removes phi expressions from a SequenceNode and its children. It also removes redundant variable assignments, e.g., *vvar_2 = vvar_1* where both vvar_1 and vvar_2 are mapped to the same variable.

**__init__**(*func*, *seq_node*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_map=None*)

**Parameters**

- **func** (Function | *str*)
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_map** (VariableMap | *None*)

**class** angr.analyses.decompiler.**Ssailification**

Bases: *Analysis*

Ssailification (SSA-AIL-ification) transforms an AIL graph to its partial-SSA form.

**__init__**(*func*, *ail_graph*, *entry=None*, *canonical_size=8*, *stack_pointer_tracker=None*, *func_addr=None*, *ail_manager=None*, *ssa_stackvars=False*, *ssa_tmps=False*, *func_args=None*, *rewrite_vvars=None*, *vvar_id_start=0*)

**Parameters**

- **func** – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **canonical_size** – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.

angr.analyses.decompiler.**StructuredCodeGenerator**

alias of *CStructuredCodeGenerator*

**class** angr.analyses.decompiler.**VariableMap**

Bases: object

A side container that maps the `.idx` of AIL `Statement` and `Expression` objects to variable-related information.

The following pieces of information are tracked:

- `variable` (a `SimVariable`) and `variable_offset` (an `int`): the variable that an AIL atom resolves to, and the offset into that variable.
- `custom_string` (a `bool`): whether a `Const` expression refers to a custom string.
- `reference_values` (a `dict` mapping `SimType` to a value): reference values associated with a `Const` expression (e.g., custom strings).
- `reference_variable` (a `SimVariable`) and `reference_variable_offset` (an `int`): the variable that a constant expression references, and the offset into it. These are siblings of `variable` / `variable_offset` that are specifically used for constants that reference global/extern variables.
- `prototype` (a `SimTypeFunction`) and `calling_convention` (a `SimCC`): the call-site prototype and calling convention associated with an AIL `Call` expression. These used to live directly on the `Call` object; they are heavy, non-serializable Python references, so they are tracked here instead.
- `variant` (an `EnumVariant`): the enum variant that a Rust `Let` expression binds.
- `returnty` (a `SimType`): the return type of a Rust `FunctionLikeMacro` call.

Keys are the integer `.idx` values of AIL Statement/Expression objects. Because `Clinic` builds one `ailment.Manager` per invocation, `.idx` values are unique within a single Clinic. So a VariableMap is scoped to one Clinic instance and is stored in the corresponding `DecompilationCache`.

**__init__**()

**variable**(*obj*)

**Return type**
*SimVariable* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**variable_offset**(*obj*)

**Return type**
int

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**custom_string**(*obj*)

**Return type**
bool

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**reference_values**(*obj*)

**Return type**
dict[*SimType*, Any] | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**reference_variable**(*obj*)

**Return type**
*SimVariable* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**reference_variable_offset**(*obj*)

**Return type**
int

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**has_variable**(*obj*)

**Return type**
bool

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**prototype**(*obj*)

**Return type**
*SimTypeFunction* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**calling_convention**(*obj*)

**Return type**
*SimCC* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**variant**(*obj*)

**Return type**
Any

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**returnty**(*obj*)

**Return type**
*SimType* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**set_variable**(*obj*, *variable*, *offset=0*)

Set the variable information for an AIL atom. If `variable` is `None`, the variable information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **variable** (SimVariable | *None*)
- **offset** (*int*)

**set_variable_offset**(*obj*, *offset*)

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **offset** (*int*)

**set_custom_string**(*obj*, *value=True*)

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **value** (*bool*)

**set_reference_values**(*obj*, *reference_values*)

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **reference_values** (*dict[SimType, Any]*)

**set_reference_variable**(*obj*, *variable*, *offset=0*)

Set the reference variable information for an AIL atom. If `variable` is `None`, the reference variable information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **variable** (SimVariable | *None*)
- **offset** (*int*)

**set_prototype**(*obj*, *prototype*)

Set the call-site prototype for an AIL Call. If `prototype` is `None`, the prototype information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **prototype** (SimTypeFunction | *None*)

**set_calling_convention**(*obj*, *cc*)

Set the calling convention for an AIL Call. If `cc` is `None`, the calling-convention information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **cc** (SimCC | *None*)

**set_variant**(*obj*, *variant*)

Set the enum variant for a Rust `Let` expression. If `variant` is `None`, the variant information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)

- **variant** (*Any*)

**set_returnty**(*obj*, *returnty*)

Set the return type for a Rust `FunctionLikeMacro` call. If `returnty` is `None`, the return-type information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **returnty** (SimType | *None*)

**transfer**(*src*, *dst*)

Copy all variable information associated with `src` to `dst`. Used when an AIL atom is deep-copied to a new `.idx` (e.g. during structuring/duplication) so that the new atom keeps the same variable association.

**Return type**
None

**Parameters**

- **src** (TaggedObject | *int*)
- **dst** (TaggedObject | *int*)

**to_json**()

Serialize this VariableMap to a JSON-compatible object.

Variables are referenced by their `.ident` (reference-by-ident); they must be resolved back to `SimVariable` objects via a resolver in `from_json()`.

**Return type**
dict[str, Any]

**classmethod from_json**(*data*, *resolve_variable*)

Deserialize a VariableMap from a JSON-compatible object produced by `to_json()`.

**Parameters**

- **data** (dict[str, Any]) – The JSON object.
- **resolve_variable** (Callable[[str], *SimVariable* | None]) – A callable that maps a variable ident (`str`) to a `SimVariable` (or `None` if it cannot be resolved).

**Return type**
*VariableMap*

## Submodules


*ail_simplifier*
*ailgraph_walker*
*block_io_finder*
*block_similarity*
*block_simplifier*
*block_walkers*
*callsite_maker*
*ccall_rewriters*


continues on next page

Table 9 – continued from previous page

| | |
|---|---|
| *clinic* | |
| *condition_processor* | |
| *counters* | |
| *decompilation_cache* | |
| *decompilation_options* | |
| *decompiler* | |
| *dephication* | |
| *dirty_rewriters* | |
| *edits* | Knowledge-base edits for decompilation output: renaming, retyping, and commenting. |
| *empty_node_remover* | |
| *expression_narrower* | |
| *goto_manager* | |
| *jump_target_collector* | |
| *jumptable_entry_condition_rewriter* | |
| *known_patterns* | Known code patterns: context-aware, declarative descriptions of library/macro idioms that compilers inline, plus machinery (KnownPatternFinder) to find them in Clinic AIL graphs and outline them back into calls. |
| *label_collector* | |
| *node_replacer* | |
| *notes* | |
| *optimization_pass_registry* | Stable name resolution for decompiler optimization passes and peephole optimizations. |
| *optimization_passes* | |
| *peephole_optimizations* | |
| *presets* | |
| *redundant_label_remover* | |
| *region_identifier* | |
| *region_overlay* | |
| *region_simplifiers* | |
| *region_walker* | |
| *return_maker* | |
| *semantic_naming* | Semantic variable naming patterns for the decompiler. |
| *seq_to_blocks* | |
| *sequence_walker* | |
| *ssailification* | |
| *stack_item* | |
| *stackarg_offset_manager* | |
| *stl_field_accessors* | Named accessors for fields of the C++ STL classes registered in the `cpp::std` `SimTypeCollection`. |
| *structured_codegen* | |
| *structurer_nodes* | |
| *structuring* | |
| *utils* | |
| *variable_map* | |

### angr.analyses.decompiler.ail_simplifier

**exception** angr.analyses.decompiler.ail_simplifier.**HasVVarNotification**

Bases: Exception

Notifies the existence of a VirtualVariable.

exception angr.analyses.decompiler.ail_simplifier.**HasRefVVarNotification**

Bases: Exception

Notifies the existence of a reference to a VirtualVariable.

class angr.analyses.decompiler.ail_simplifier.**AILBlockTempCollector**

Bases: *AILBlockViewer*

Collects any temporaries used in a block.

**__init__**(***kwargs*)

class angr.analyses.decompiler.ail_simplifier.**DefEqRelation**

Bases: Enum

Describes the location relationship between a virtual variable definition and the equivalence statement.

**UNKNOWN = 0**

**DEF_IS_FUNCARG = 1**

**DEF_EQ_SAME_BLOCK = 2**

**DEF_IN_EQ_PRED_BLOCK = 3**

class angr.analyses.decompiler.ail_simplifier.**PartialConstantExprRewriter**

Bases: *AILBlockRewriter*

Rewrites expressions whose high bits are definitely zero to constants (if possible) or mask them with masks properly.

**__init__**(*varid*, *zero_high_bits*)

**Parameters**

- **varid** (*int*)
- **zero_high_bits** (*int*)

class angr.analyses.decompiler.ail_simplifier.**InsertLowBitsReadRewriter**

Bases: *AILBlockRewriter*

Rewrites truncating reads of Insert definitions: given `vvar = Insert(base, 0, value)`, the use `Convert(vvar.bits->value.bits, vvar)` reads back exactly the inserted value.

**__init__**(*varid*, *value_expr*)

**Parameters**

- **varid** (*int*)
- **value_expr** (Expression)

class angr.analyses.decompiler.ail_simplifier.**AILSimplifier**

Bases: *Analysis*

Perform function-level simplifications.

**__init__**(*func*, *func_graph*, *ail_manager*, *remove_dead_memdefs=False*, *stackarg_offset_manager=None*, *unify_variables=False*, *gp=None*, *narrow_expressions=False*, *fold_expressions=True*, *only_consts=False*, *fold_callexprs_into_conditions=False*, *use_callee_saved_regs_at_return=True*, *rewrite_ccalls=True*, *rename_ccalls=True*, *rewrite_dirty=True*, *removed_vvar_ids=None*, *arg_vvars=None*, *avoid_vvar_ids=None*)

**blocks:** dict[Block, Block]

**simplified:** bool

**simplified_blocks:** set[tuple[int, int | None]]

### angr.analyses.decompiler.ailgraph_walker

**exception** angr.analyses.decompiler.ailgraph_walker.**RemoveNodeNotice**

Bases: Exception

**class** angr.analyses.decompiler.ailgraph_walker.**AILGraphWalker**

Bases: object

Walks an AIL graph and optionally replaces each node with a new node.

**__init__**(*graph*, *handler*, *replace_nodes=False*, *strict_order_start=()*)

**walk**()

angr.analyses.decompiler.ailgraph_walker.**traverse_in_order**(*ail_graph*, *entry_blocks*, *visitor*)

**Parameters**

- **ail_graph** (*networkx.DiGraph[ailment.Block]*)
- **entry_blocks** (*Sequence[ailment.Block]*)
- **visitor** (Callable*[[ailment.Block], Any]*)

### angr.analyses.decompiler.block_io_finder

**class** angr.analyses.decompiler.block_io_finder.**BlockIOFinder**

Bases: *AILBlockViewer*

Finds the input and output locations of each statement in an AIL block. I/O locations can be a Register, MemoryLocation, or SpOffset (wrapped in a Memory Location).

**__init__**(*ail_obj*, *project*, *as_atom=True*)

**Parameters**
**ail_obj** (*Block | list[Statement]*)

**can_swap**(*stmt*, *ail_obj*, *offset*)

**Parameters**

- **ail_obj** (*Block | list[Statement]*)
- **offset** (*int*)

### angr.analyses.decompiler.block_similarity

angr.analyses.decompiler.block_similarity.**has_similar_stmt**(*blk1*, *blk2*)

Returns True if blk1 has a statement that is similar to a statement in blk2, False otherwise.

**Parameters**

- **blk1** (*Block*)
- **blk2** (*Block*)

angr.analyses.decompiler.block_similarity.**is_similar**(*ail_obj1*, *ail_obj2*, *graph=None*, *partial=True*, *max_depth=10*, *curr_depth=0*)

Returns True if the two AIL objects are similar, False otherwise.

**Parameters**

- **ail_obj1** (*Block | Statement*)
- **ail_obj2** (*Block | Statement*)
- **graph** (*DiGraph | None*)
- **partial** (*bool*)
- **max_depth** (*int*)
- **curr_depth** (*int*)

angr.analyses.decompiler.block_similarity.**index_of_similar_stmts**(*search_stmts*, *other_stmts*, *graph=None*, *all_positions=False*)

Returns the index of the first occurrence of the search_stmts (a list of Statement) in other_stmts (a list of Statement). If all_positions is True, returns a list of all positions.

@return: None or int (position start in other)

**Return type**
int | None

angr.analyses.decompiler.block_similarity.**in_other**(*stmts*, *other*, *graph=None*)

Returns True if the stmts (a list of Statement) is found as a subsequence in other

@return:

angr.analyses.decompiler.block_similarity.**longest_ail_subseq**(*stmts_list*, *graph=None*)

Given a list of List[Statement], it returns the longest List[Statement] that is a subsequence of all the lists. The common List[Statement] most all be in the same order and adjacent to each other. If no common subsequence is found, it returns None.

**Parameters**

- **stmts_list** (list[list[*Statement*]])
- **graph**

**Return type**
tuple[list[*Statement*] | None, list[int] | None]

**Returns**
Tuple[List[Statement], List[int]], where the first element is the longest common subsequence, and the second element is a list of integers indicating the index of the longest common subsequence in each list of statements.

### angr.analyses.decompiler.block_simplifier

**class** angr.analyses.decompiler.block_simplifier.**PeepholeOptimizationBundle**

Bases: object

PeepholeOptimizationBundle describes a set of initialized peephole optimizer instances and the dispatch structures derived from them. This bundle of peephole optimizations is reusable across *BlockSimplifier* invocations (so we avoid rebuilding the same optimizer instances).

__init__(*project*, *kb*, *ail_manager*, *func_addr=None*, *preserve_vvar_ids=None*, *type_hints=None*, *peephole_optimizations=None*)

**Parameters**

- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)
- **peephole_optimizations** (*Iterable[type[PeepholeOptimizationStmtBase | PeepholeOptimizationExprBase | PeepholeOptimizationMultiStmtBase]] | None*)

**expr_opts**

**stmt_opts**

**multistmt_opts**

**stmt_opts_by_kind**

**expr_walker**

**matches**(*project*, *ail_manager*, *func_addr*, *preserve_vvar_ids*, *type_hints*, *peephole_optimizations*)

**Return type**
bool

**Parameters**

- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list | None*)

class angr.analyses.decompiler.block_simplifier.**BlockSimplifier**

Bases: object

Simplify an AIL block.

Deliberately not an Analysis: it is instantiated once per block, hundreds of times per decompilation, so it skips the analysis-factory ceremony. Instantiate it directly with the project as the first argument; exceptions always propagate.

__init__(*project*, *block*, *ail_manager*, *func_addr=None*, *stack_pointer_tracker=None*, *peephole_optimizations=None*, *preserve_vvar_ids=None*, *type_hints=None*, *cached_reaching_definitions=None*, *cached_propagator=None*, *peephole_bundle=None*)

**Parameters**

- **block** (Block | None) – The AIL block to simplify. Setting it to None to skip calling self._analyze(), which is useful in test cases.
- **peephole_bundle** (*PeepholeOptimizationBundle* | None) – A pre-built PeepholeOptimizationBundle to reuse. Its construction parameters must match this BlockSimplifier's; callers that simplify many blocks should build one bundle and pass it to every BlockSimplifier they create.

- **project** (Project)
- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **peephole_optimizations** (*Iterable[type[PeepholeOptimizationStmtBase | PeepholeOptimizationExprBase | PeepholeOptimizationMultiStmtBase]] | None*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)

**static replace_and_build**(*block*, *replacements*, *ail_manager*, *replace_assignment_dsts=False*, *replace_loads=False*, *gp=None*, *replace_registers=True*, *max_expr_depth=13*, *max_stmt_size=20000*)

**Return type**
tuple[bool, Block]

**Parameters**

- **block** (*Block*)
- **replacements** (*Mapping[AILCodeLocation, Mapping[Expression, Expression]]*)
- **ail_manager** (*Manager*)
- **replace_assignment_dsts** (*bool*)
- **replace_loads** (*bool*)
- **gp** (*int | None*)
- **replace_registers** (*bool*)
- **max_expr_depth** (*int | None*)
- **max_stmt_size** (*int*)

### angr.analyses.decompiler.block_walkers

**exception** angr.analyses.decompiler.block_walkers.**HasCallNotification**

Bases: Exception

Abort the walk on the first Call / SideEffectStatement encountered.

**class** angr.analyses.decompiler.block_walkers.**HasCallExprWalker**

Bases: *AILBlockViewer*

Singleton walker that raises HasCallNotification on the first Call / SideEffectStatement it visits.

### angr.analyses.decompiler.callsite_maker

**class** angr.analyses.decompiler.callsite_maker.**CallSiteMaker**

Bases: object

Add calling convention, declaration, and args to a call site.

__init__(*project*, *block*, *, *ail_manager*, *reaching_definitions=None*, *stack_pointer_tracker=None*)

**Parameters**

- **project** (Project)
- **ail_manager** (*Manager*)
- **reaching_definitions** (SRDAModel | *None*)

### angr.analyses.decompiler.ccall_rewriters

### Submodules


*amd64_ccalls*
*arm_ccalls*
*rewriter_base*
*x86_ccalls*


### angr.analyses.decompiler.ccall_rewriters.amd64_ccalls

class angr.analyses.decompiler.ccall_rewriters.amd64_ccalls.**AMD64CCallRewriter**

Bases: *CCallRewriterBase*

Implements VEX ccall rewriter for AMD64.

### angr.analyses.decompiler.ccall_rewriters.arm_ccalls

class angr.analyses.decompiler.ccall_rewriters.arm_ccalls.**ARMCCallRewriter**

Bases: *CCallRewriterBase*

Rewrites ARM flag helpers following libVEX's guest_arm_helpers.c. cond_n_op packs the condition code in bits 7:4 and the cc_op in bits 3:0; odd condition codes negate their even partner.

Flag semantics per cc_op (dep_1, dep_2, dep_3):

COPY: NZCV in bits 31:28 of dep_1 ADD: res = dep_1 + dep_2; C = res <u dep_1 SUB: res = dep_1 - dep_2; C = dep_1 >=u dep_2 ADC: res = dep_1 + dep_2 + dep_3; C = res <=u dep_1 if dep_3 else res <u dep_1 SBB: res = dep_1 - dep_2 - (dep_3^1); C = dep_1 >=u dep_2 if dep_3 else dep_1 >u dep_2 LOGIC: res = dep_1; C = dep_2 (shifter carry out); V = dep_3 (old V) MUL: res = dep_1; C = dep_3[1]; V = dep_3[0] MULL: res = dep_2:dep_1; C = dep_3[1]; V = dep_3[0]

ADC with dep_3 == 0 is ADD and SBB with dep_3 == 1 is SUB. Only shapes that map exactly onto AIL expressions are rewritten; everything else is left alone.

### angr.analyses.decompiler.ccall_rewriters.rewriter_base

class angr.analyses.decompiler.ccall_rewriters.rewriter_base.**CCallRewriterBase**

Bases: object

The base class for CCall rewriters.

__init__(*ccall*, *project*, *ail_manager*, *rename_ccalls=False*)

**Parameters**

- **ccall** (VEXCCallExpression)
- **ail_manager** (*Manager*)

- **rename_ccalls** (*bool*)

**project**

**ail_manager**

**result:** *Expression* | None

### angr.analyses.decompiler.ccall_rewriters.x86_ccalls

**class** angr.analyses.decompiler.ccall_rewriters.x86_ccalls.**X86CCallRewriter**

Bases: *CCallRewriterBase*

Implements VEX ccall rewriter for X86.

From libVEX, a summary of the field usages is:

```
Operation          DEP1               DEP2               NDEP
-------------------------------------------------------------
add/sub/mul        first arg          second arg         unused
adc/sbb            first arg          (second arg)
                                      XOR old_carry      old_carry
and/or/xor         result             zero               unused
inc/dec            result             zero               old_carry
shl/shr/sar        result             subshifted-        unused
                                      result
rol/ror            result             zero               old_flags
copy               old_flags          zero               unused.
```

### angr.analyses.decompiler.clinic

**class** angr.analyses.decompiler.clinic.**BlockCache**

Bases: tuple

BlockCache(rd, prop)

**static __new__**(*_cls*, *rd*, *prop*)

Create new instance of BlockCache(rd, prop)

**prop**

Alias for field number 1

**rd**

Alias for field number 0

**class** angr.analyses.decompiler.clinic.**ClinicMode**

Bases: Enum

Analysis mode for Clinic.

**DECOMPILE = 1**

**COLLECT_DATA_REFS = 2**

**class** angr.analyses.decompiler.clinic.**DataRefDesc**

Bases: object

The fields of this class is compatible with items inside IRSB.data_refs.

**data_addr:** int

**data_size:** int

**block_addr:** int

**stmt_idx:** int

**ins_addr:** int

**data_type_str:** str

**__init__**(*data_addr*, *data_size*, *block_addr*, *stmt_idx*, *ins_addr*, *data_type_str*)

**Parameters**

- **data_addr** (*int*)
- **data_size** (*int*)
- **block_addr** (*int*)
- **stmt_idx** (*int*)
- **ins_addr** (*int*)
- **data_type_str** (*str*)

**Return type**
None

**class** angr.analyses.decompiler.clinic.**ClinicStage**

Bases: IntEnum

Different stages of treating an ailment.

**INITIALIZATION = 0**

**AIL_GRAPH_CONVERSION = 1**

**MAKE_RETURN_SITES = 2**

**MAKE_ARGUMENT_LIST = 3**

**TRACK_STACK_POINTERS = 4**

**CONSTANT_PROPAGATION = 5**

**MAKE_CALLSITES = 6**

**POST_CALLSITES = 7**

**PRE_SSA_LEVEL0_FIXUPS = 8**

**SSA_LEVEL0_TRANSFORMATION = 9**

**PRE_SSA_LEVEL1_SIMPLIFICATIONS = 10**

**SSA_LEVEL1_TRANSFORMATION = 11**

**POST_SSA_LEVEL1_SIMPLIFICATIONS = 12**

**RECOVER_VARIABLES = 13**

**SEMANTIC_VARIABLE_NAMING = 14**

**COLLECT_EXTERNS = 15**

**__new__**(*value*)

**class** angr.analyses.decompiler.clinic.**ComboRegReferenceWalker**

Bases: *AILBlockRewriter*

Rewrite references to combo registers to load from the combo register.

**__init__**(*project*, *ail_manager*)

**class** angr.analyses.decompiler.clinic.**Clinic**

Bases: *Analysis*, *Serializable*

A Clinic deals with AILments: it lifts a function to AIL and runs the decompiler's simplification pipeline on it.

AIL graphs exposed after a DECOMPILE-mode run:

- `cc_graph`: the graph at the end of Clinic's own simplification, frozen as a copy before region identification. Serialized; available on both live and cached-and-reloaded clinics.
- `graph`: `cc_graph` further transformed by the decompiler's graph-simplification passes, region identification, and region-simplification passes — the final graph. Serialized; available on both live and cached-and-reloaded clinics.
- `unoptimized_graph`: a copy taken before the first structure-altering optimization pass; use it for an exact instruction-to-AIL mapping. Only built (and serialized) with `Decompiler(save_unoptimized_graph=True)`; otherwise it is None on both live and cached-and-reloaded clinics.
- `_ail_graph` / `_init_ail_graph`: pipeline internals; never serialized.

**__init__**(*func*, *remove_dead_memdefs=False*, *exception_edges=False*, *sp_tracker_track_memory=True*, *fold_expressions=True*, *fold_callexprs_into_conditions=False*, *insert_labels=True*, *optimization_passes=None*, *cfg=None*, *peephole_optimizations=None*, *must_struct=None*, *reset_variable_names=False*, *rewrite_ites_to_diamonds=True*, *rewrite_ites_to_diamond_max_cases=15*, *cache=None*, *mode=ClinicMode.DECOMPILE*, *sp_shift=0*, *inline_functions=None*, *inlined_counts=None*, *inlining_parents=None*, *vvar_id_start=0*, *optimization_scratch=None*, *desired_variables=None*, *force_loop_single_exit=True*, *refine_loops_with_single_successor=False*, *expose_loop_head_backedges=False*, *typehoon_cls=<class 'angr.analyses.typehoon.typehoon.Typehoon'>*, *max_ail_statements=1000000*, *max_type_constraints=100000*, *type_constraint_set_degradation_threshold=150*, *ail_graph=None*, *arg_vvars=None*, *start_stage=ClinicStage.INITIALIZATION*, *end_stage=None*, *skip_stages=()*, *notes=None*, *static_vvars=None*, *static_buffers=None*, *flatten_args=False*, *constrain_callee_prototypes=False*, *semvar_naming=True*, *flavor='pseudocode'*, *variable_map=None*, *save_unoptimized_graph=False*, *known_patterns=None*, *recognize_known_patterns=True*)

**Parameters**

- **func** (*Function*)
- **peephole_optimizations** (*Iterable[type[PeepholeOptimizationStmtBase | PeepholeOptimizationExprBase]] | None*)
- **must_struct** (*set[str] | None*)
- **rewrite_ites_to_diamond_max_cases** (*int*)

- **cache** (DecompilationCache | *None*)
- **mode** (ClinicMode)
- **sp_shift** (*int*)
- **inline_functions** (*set[Function] | None*)
- **inlined_counts** (*dict[int, int] | None*)
- **inlining_parents** (*set[int] | None*)
- **vvar_id_start** (*int*)
- **optimization_scratch** (*dict[str, Any] | None*)
- **desired_variables** (*set[str] | None*)
- **force_loop_single_exit** (*bool*)
- **refine_loops_with_single_successor** (*bool*)
- **expose_loop_head_backedges** (*bool*)
- **max_ail_statements** (*int*)
- **max_type_constraints** (*int*)
- **type_constraint_set_degradation_threshold** (*int*)
- **ail_graph** (*DiGraph | None*)
- **arg_vvars** (*dict[int, tuple[VirtualVariable, SimVariable]] | None*)
- **start_stage** (ClinicStage | *None*)
- **end_stage** (ClinicStage | *None*)
- **skip_stages** (*tuple[ClinicStage, ...]*)
- **notes** (*dict[str, DecompilationNote] | None*)
- **static_vvars** (*dict | None*)
- **static_buffers** (*dict | None*)
- **constrain_callee_prototypes** (*bool*)
- **semvar_naming** (*bool*)
- **flavor** (*str*)
- **variable_map** (VariableMap | *None*)
- **save_unoptimized_graph** (*bool*)
- **known_patterns** (*str | tuple[str, ...] | None*)
- **recognize_known_patterns** (*bool*)

**cc_graph:** DiGraph | None

**unoptimized_graph:** DiGraph | None

**arg_vvars:** dict[int, tuple[*VirtualVariable*, *SimVariable*]] | None

**variable_map:** *VariableMap*

**externs:** set[*SimMemoryVariable*]

**data_refs:** dict[int, list[*DataRefDesc*]]

**entry_node_addr:** Address

**unresolvable_peephole_optimizations:** list[str]

**reaching_definitions:** *SRDAModel* | None

**vvar_to_vvar:** dict[int, int] | None

**secondary_stackvars:** set[int]

**edges_to_remove:** list[tuple[Address, Address]]

**copied_var_ids:** set[int]

**typehoon:** *Typehoon* | None

**stack_items:** dict[int, *StackItem*]

**block**(*addr*, *size*)

Get the converted block at the given specific address with the given size.

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**Returns**

**dbg_repr**()

**Returns**

**calculate_stack_depth**()

**copy_graph**(*graph=None*)

**Return type**
DiGraph

**parse_variable_addr**(*addr*)

**Return type**
tuple[Any, Any]

**Parameters**
**addr** (Expression)

**new_block_addr**()

Return a block address that does not conflict with any existing blocks.

**Return type**
int

**Returns**
The block address.

**static remove_empty_nodes**(*graph*)

**Return type**
DiGraph

**Parameters**
**graph** (*DiGraph*)

**constrain_callee_prototypes**()

Constrain the types of callee function arguments based on facts that are observed at call sites. Note that this function will change the prototypes of (callee) functions in the knowledge base, which means it may affect the decompilation output of the current function if it is decompiled again.

**resolve_peephole_optimizations**()

Retry resolving peephole-optimization names that were unresolvable at parse time (their defining module may have been imported since). Resolved classes move into `peephole_optimizations`; names that still do not resolve stay in `unresolvable_peephole_optimizations`.

**Return type**
None

**classmethod parse_from_cmessage**(*cmsg*, *, *project=None*, *kb=None*, *function=None*, *cfg=None*, ***kwargs*)

Bypasses *Clinic.__init__()* (which runs the analysis) and reconstructs the instance directly. Runtime back-references — project / kb / function / _cfg — come from kwargs.

Only the state consumed by the decompiler's cache-reuse path and by post-decompilation consumers (cc_graph, graph, optionally unoptimized_graph) is serialized; regenerable and runtime-only state is restored to its default here. If `function` is None and `kb` is provided, the function is resolved by address from the cmessage.

### angr.analyses.decompiler.condition_processor

**class** angr.analyses.decompiler.condition_processor.**AILExprIdAnnotation**

Bases: Annotation

An annotation that we use to annotate BVVs so that they are differentiable between other BVVs with the same value and size.

**eliminatable = True**

**relocatable = False**

**class** angr.analyses.decompiler.condition_processor.**ConditionProcessor**

Bases: object

Convert between claripy AST and AIL expressions. Also calculates reaching conditions of all nodes on a graph.

**__init__**(*arch*, *ail_manager*, *condition_mapping=None*)

**Parameters**
**ail_manager** (*Manager*)

**jump_table_conds: dict[int, set]**

**clear**()

**have_opposite_edge_conditions**(*graph*, *src*, *dst0*, *dst1*)

Check if the edge conditions of two edges (src, dst0) and (src, dst1) are opposite to each other. Try to avoid condition translation if possible.

**Return type**
bool

**Parameters**
**graph** (*DiGraph*)

**recover_edge_condition**(*graph*, *src*, *dst*)

**Parameters**
**graph** (*DiGraph*)

**recover_edge_conditions**(*region*, *graph=None*)

**Return type**
dict

**recover_reaching_conditions**(*region*, *graph=None*, *with_successors=False*, *case_entry_to_switch_head=None*, *simplify_conditions=True*)

Recover the reaching conditions for each block in an acyclic graph. Note that we assume the graph that's passed in is acyclic.

**Parameters**

- **case_entry_to_switch_head** (*dict[int, int] | None*)
- **simplify_conditions** (*bool*)

**remove_claripy_bool_asts**(*node*, *memo=None*)

**classmethod get_last_statement**(*block*)

This is the buggy version of get_last_statements, because, you know, there can always be more than one last statement due to the existence of branching statements (like, If-then-else). All methods using get_last_statement() should switch to get_last_statements() and properly handle multiple last statements.

**classmethod get_last_statements**(*block*)

**Return type**
list[*Statement* | *ConditionalBreakNode* | *BreakNode* | *ContinueNode* | None]

**EXC_COUNTER = 1000**

**convert_claripy_bool_ast**(*cond*, *memo=None*)

Convert recovered reaching conditions from claripy ASTs to ailment Expressions

**Returns**
None

**convert_claripy_bool_ast_core**(*cond*, *memo*)

**claripy_ast_from_ail_condition**(*condition*, *, *nobool=False*, *must_bool=False*, *ins_addr=0*)

**Return type**
Bool | Bits

**Parameters**

- **nobool** (*bool*)
- **must_bool** (*bool*)

- **ins_addr** (*int*)

**static claripy_ast_to_sympy_expr**(*ast*, *memo=None*)

**static sympy_expr_to_claripy_ast**(*expr*, *memo*)

**Parameters**
**memo** (*dict*)

**static simplify_condition**(*cond*, *depth_limit=8*, *variables_limit=8*)

**create_jump_target_var**(*jumptable_head_addr*)

**Parameters**
**jumptable_head_addr** (*int*)

### angr.analyses.decompiler.counters

**class** angr.analyses.decompiler.counters.**AILBlockCallCounter**

Bases: *AILBlockViewer*

Helper class to count AIL calls and call-expressions in a block, or collect call statements and call expressions as well as their locations.

**__init__**(*consider_conditions=False*)

**Parameters**
**consider_conditions** (*bool*)

**class** angr.analyses.decompiler.counters.**BooleanCounter**

Bases: *AILBlockViewer*

This class counts the number of Boolean operators an expression has. In the case of: *if (a || (b && c))*, it will count 2 Boolean operators.

**__init__**()

**class** angr.analyses.decompiler.counters.**ControlFlowStructureCounter**

Bases: *SequenceWalker*

Counts the number of different types of control flow structures found in a sequence of nodes. This should be used after the sequence has been simplified.

**__init__**(*node*)

**class** angr.analyses.decompiler.counters.**OperatorCounter**

Bases: *AILBlockViewer*

Count the occurrence of a given expression operator.

**__init__**(*operator*, *expr_or_stmt*)

**Parameters**

- **operator** (*str | Iterable[str]*)
- **expr_or_stmt** (*Expression | Statement*)

**class** angr.analyses.decompiler.counters.**RegisterExpressionCounter**

Bases: *AILBlockViewer*

Count the occurrence of all register expressions in expr

__init__(*expr_or_stmt*)

**Parameters**
**expr_or_stmt** (Expression | Statement)

class angr.analyses.decompiler.counters.**SingleExpressionCounter**

Bases: *AILBlockViewer*

Count the occurrence of subexpr in expr.

__init__(*stmt*, *subexpr*)

### Submodules


*boolean_counter*
*call_counter*
*expression_counters*
*seq_cf_structure_counter*


### angr.analyses.decompiler.counters.boolean_counter

class angr.analyses.decompiler.counters.boolean_counter.**BooleanCounter**

Bases: *AILBlockViewer*

This class counts the number of Boolean operators an expression has. In the case of: *if (a || (b && c))*, it will count 2 Boolean operators.

__init__()

### angr.analyses.decompiler.counters.call_counter

class angr.analyses.decompiler.counters.call_counter.**AILBlockCallCounter**

Bases: *AILBlockViewer*

Helper class to count AIL calls and call-expressions in a block, or collect call statements and call expressions as well as their locations.

__init__(*consider_conditions=False*)

**Parameters**
**consider_conditions** (*bool*)

class angr.analyses.decompiler.counters.call_counter.**AILCallCounter**

Bases: *SequenceWalker*

Helper class to count AIL calls and call expressions in a structuring node, or collect call statements and call expressions as well as their locations.

__init__(*consider_conditions=False*)

**Parameters**
**consider_conditions** (*bool*)

### angr.analyses.decompiler.counters.expression_counters

class angr.analyses.decompiler.counters.expression_counters.**SingleExpressionCounter**

Bases: *AILBlockViewer*

Count the occurrence of subexpr in expr.

**__init__**(*stmt*, *subexpr*)

class angr.analyses.decompiler.counters.expression_counters.**RegisterExpressionCounter**

Bases: *AILBlockViewer*

Count the occurrence of all register expressions in expr

**__init__**(*expr_or_stmt*)

**Parameters**

**expr_or_stmt** (Expression | Statement)

class angr.analyses.decompiler.counters.expression_counters.**OperatorCounter**

Bases: *AILBlockViewer*

Count the occurrence of a given expression operator.

**__init__**(*operator*, *expr_or_stmt*)

**Parameters**

- **operator** (*str* | *Iterable[str]*)
- **expr_or_stmt** (Expression | Statement)

### angr.analyses.decompiler.counters.seq_cf_structure_counter

class angr.analyses.decompiler.counters.seq_cf_structure_counter.**ControlFlowStructureCounter**

Bases: *SequenceWalker*

Counts the number of different types of control flow structures found in a sequence of nodes. This should be used after the sequence has been simplified.

**__init__**(*node*)

### angr.analyses.decompiler.decompilation_cache

class angr.analyses.decompiler.decompilation_cache.**DecompilationCache**

Bases: *Serializable*

Caches key data structures that can be used later for refining decompilation results, such as retyping variables.

**__init__**(*addr*)

**parameters:** dict[str, Any]

**version:** str

**timestamp:** int

**addr**

**cfg:** *CFGModel* | None

**type_constraints:** dict[*TypeVariable*, set[*TypeConstraint*]]

**arg_vvars:** dict

**func_typevar:** *TypeVariable* | None

**var_to_typevar:** dict

**stackvar_max_sizes:** dict

**stack_offset_typevars:** dict

**codegen:** *BaseStructuredCodeGenerator* | None

**clinic:** *Clinic* | None

**variable_map:** *VariableMap* | None

**ite_exprs:** set[tuple[int, *Expression*]]

**binop_operators:** dict[*OpDescriptor*, str]

**errors:** list[str]

**function_summary:** str | None

**notes:** dict[str, *DecompilationNote*]

**max_tv_id:** int

**property local_types**

**classmethod parse_from_cmessage**(*cmsg*, *, *project=None*, *kb=None*, *function=None*, *cfg=None*, **_)

Parse a DecompilationCache from a cmessage. Runtime back-references (project, kb, function, cfg) are passed through to the embedded Clinic / codegen parsers so the parsed cache is functional for cache-hit validity checks. Decompilation variables live on kb.dec_variables.

### angr.analyses.decompiler.decompilation_options

**class** angr.analyses.decompiler.decompilation_options.**DecompilationOption**

Bases: Generic

Describes a decompilation option.

**__init__**(*name*, *description*, *value_type*, *cls*, *param*, *value_range=None*, *category='General'*, *default_value=None*, *clears_cache=True*, *candidate_values=None*, *convert=None*)

**Parameters**

- **name** (*str*)
- **description** (*str*)
- **value_type** (type*[T]*)
- **cls** (*str*)
- **param** (*str*)
- **category** (*str*)
- **default_value** (*T | None*)

- **clears_cache** (*bool*)
- **candidate_values** (*list[T] | None*)
- **convert** (*Callable[[str], T] | None*)

angr.analyses.decompiler.decompilation_options.**O**

alias of *DecompilationOption*

angr.analyses.decompiler.decompilation_options.**parse_known_patterns**(*value*)

Normalize the `known_patterns` option value into `None` (force-enable nothing), the string `"all"`, or a tuple of pattern names.

Both spellings are accepted so that the GUI can offer a single text field (`"all" / "IsListEmpty, std::string::front"`) while API callers can pass a list. Names are not resolved here; that happens against the template registry, which is where an unknown name is reported.

**Return type**
`str | tuple[str, ...] | None`

angr.analyses.decompiler.decompilation_options.**get_structurer_option**()

**Return type**
*DecompilationOption* | None

### angr.analyses.decompiler.decompiler

**class** angr.analyses.decompiler.decompiler.**ConstantCollector**

Bases: *SequenceWalker*

Collect the values of all integer constants in a structured sequence, including the constants inside expressions and statements that structuring lifts out of blocks and onto the nodes themselves.

**__init__**()

**class** angr.analyses.decompiler.decompiler.**Decompiler**

Bases: *Analysis*

The decompiler analysis.

Run this on a Function object for which a normalized CFG has been constructed. The fully processed output can be found in result.codegen.text

AIL graphs exposed on the result (both on a fresh run and on a cache hit, including caches reloaded from angrdb or the runtime-db spill):

- `ail_graph` (= `clinic.cc_graph`): the simplified graph before region identification.
- `clinic.graph`: the final graph after region identification and region simplification.
- `unoptimized_ail_graph` (= `clinic.unoptimized_graph`): a snapshot before the first structure-altering optimization pass; use it for an exact instruction-to-AIL mapping. Only built when `save_unoptimized_graph=True` is passed; otherwise this attribute is None on both fresh runs and cache hits.

Complexity guard: pathologically large functions (e.g. packer filler that linear disassembly turns into thousands of blocks) can keep the pipeline busy indefinitely, so two size limits are checked before the expensive work starts. `max_function_blocks` (default 50000, `DEFAULT_MAX_FUNCTION_BLOCKS`) bounds the number of CFG basic blocks and is checked before any AIL is built; `max_ail_statements` (default 1000000, `DEFAULT_MAX_AIL_STATEMENTS`) bounds the number of AIL statements and is checked right after the AIL graph is built. Both are ~10x the largest function in angr's test binary corpus. Either limit can be changed through the decompilation option of the same name, and setting one to 0 disables it. When a limit is exceeded,

decompilation stops with an `AngrDecompilationComplexityError` naming the limit and the actual size: with `fail_fast=True` it is raised, otherwise it is recorded in `self.errors` (and in `kb.decompilations[...].errors`) and exposed as *complexity_error*, and `codegen` stays None.

The optimization passes come from `preset` (a name or a DecompilationPreset), or from an explicit `optimization_passes` list when no preset is given. `disable_opts` drops individual passes from the preset's list, which is how a caller asks for "this preset, minus that one pass" without restating it.

**__init__**(*func*, *cfg=None*, *options=None*, *preset=None*, *optimization_passes=None*, *disable_opts=None*, *sp_tracker_track_memory=True*, *peephole_optimizations=None*, *vars_must_struct=None*, *flavor='pseudocode'*, *expr_comments=None*, *stmt_comments=None*, *ite_exprs=None*, *binop_operators=None*, *decompile=True*, *regen_clinic=False*, *inline_functions=None*, *desired_variables=None*, *update_memory_data=True*, *want_full_graph=False*, *generate_code=True*, *use_cache=True*, *update_cache=True*, *expr_collapse_depth=16*, *clinic_graph=None*, *clinic_arg_vvars=None*, *clinic_start_stage=None*, *clinic_end_stage=None*, *clinic_skip_stages=()*, *static_vvars=None*, *static_buffers=None*, *codegen_cls=<class 'angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator'>*, *save_unoptimized_graph=False*)

**Parameters**

- **func** (Function | *str* | *int*)
- **cfg** (CFGFast | CFGModel | *None*)
- **preset** (*str* | DecompilationPreset | *None*)
- **peephole_optimizations** (*Iterable*[type[PeepholeOptimizationStmtBase] | type[PeepholeOptimizationExprBase]] | *None*)
- **vars_must_struct** (*set*[*str*] | *None*)
- **update_memory_data** (*bool*)
- **want_full_graph** (*bool*)
- **generate_code** (*bool*)
- **use_cache** (*bool*)
- **update_cache** (*bool*)
- **expr_collapse_depth** (*int*)
- **static_vvars** (*dict* | *None*)
- **static_buffers** (*dict* | *None*)
- **save_unoptimized_graph** (*bool*)

**property complexity_error:** *AngrDecompilationComplexityError* | None

The complexity-limit violation that aborted this decompilation, or None if no limit was exceeded. Inspect `limit_name`, `limit`, and `actual` on it to decide whether to raise the limit or skip the function.

**reflow_variable_types**(*cache*)

Re-run type inference on an existing variable recovery result, then rerun codegen to generate new results.

**Returns**

**Parameters**

**cache** (DecompilationCache)

**find_data_references_and_update_memory_data**(*seq_node*)

**Parameters**
**seq_node** (SequenceNode)

**transform_graph_from_ssa**(*ail_graph*)

Translate an SSA AIL graph out of SSA form. This is useful for producing a non-SSA AIL graph for displaying in angr management.

**Parameters**
**ail_graph** (DiGraph) – The AIL graph to transform out of SSA form.

**Return type**
DiGraph

**Returns**
The translated AIL graph.

**transform_seqnode_from_ssa**(*seq_node*)

**Return type**
*SequenceNode*

**Parameters**
**seq_node** (SequenceNode)

**llm_refine**()

Use the configured LLM to suggest improved variable names, function names, and variable types. Returns True if any changes were made.

**Return type**
bool

**llm_suggest_variable_names**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest better variable names for the decompiled code. Returns True if any variables were renamed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller. If False (default), exceptions are caught and the method returns False.
- **code_text** (*str | None*)

**Return type**
bool

**llm_suggest_function_name**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest a better function name. Only suggests rename for auto-generated names (starting with sub_ or fcn.). Returns True if the function was renamed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
bool

**llm_suggest_variable_types**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to suggest better C types for variables. Returns True if any variable types were changed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
bool

**llm_summarize_function**(*llm_client=None*, *code_text=None*, *raise_exc=False*)

Ask the LLM to produce a natural-language summary of what the decompiled function does. The summary is stored in the DecompilationCache and returned.

Returns the summary string, or None if summarization failed.

**Parameters**

- **raise_exc** (bool) – If True, exceptions from the LLM call are propagated to the caller.
- **code_text** (*str | None*)

**Return type**
str | None

**static options_to_params**(*options*)

Convert decompilation options to a dict of params.

**Parameters**
**options** (list[tuple[*DecompilationOption*, Any]]) – The decompilation options.

**Return type**
dict[str, Any]

**Returns**
A dict of keyword arguments.

### angr.analyses.decompiler.dephication

**class** angr.analyses.decompiler.dephication.**GraphDephication**

Bases: *DephicationBase*

GraphDephication removes phi expressions from an AIL graph, essentially transforms a partial-SSA form of AIL graph to a normal AIL graph.

**__init__**(*func*, *ail_graph*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_map=None*)

**Parameters**

- **func** (*Function* | str) – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_map** (VariableMap | *None*)

**class** angr.analyses.decompiler.dephication.**GraphDephicationVVarMapping**

Bases: *Analysis*

GraphDephicationVVarMapping collects information from an AIL graph regarding how virtual variables should be mapped to each other in order to remove phi variables. This mapping will later be used to transform a partial-SSA form of AIL graph to a conventional AIL graph.

This class largely implements "Translating Out of Static Single Assignment Form" by Sreedhar et al.

**__init__**(*func*, *ail_graph*, *ail_manager*, *entry=None*, *vvar_id_start=0*, *arg_vvars=None*)

**Parameters**

- **func** (*Function* | str) – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **ail_manager** (*Manager*)
- **vvar_id_start** (*int*)
- **arg_vvars** (*list[VirtualVariable] | None*)

**class** angr.analyses.decompiler.dephication.**SeqNodeDephication**

Bases: *DephicationBase*

SeqNodeDephication removes phi expressions from a SequenceNode and its children. It also removes redundant variable assignments, e.g., *vvar_2 = vvar_1* where both vvar_1 and vvar_2 are mapped to the same variable.

**__init__**(*func*, *seq_node*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_map=None*)

**Parameters**

- **func** (*Function | str*)
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_map** (*VariableMap | None*)

## Submodules


*dephication_base*
*graph_dephication*
*graph_rewriting*
*graph_vvar_mapping*
*rewriting_engine*
*seqnode_dephication*


### angr.analyses.decompiler.dephication.dephication_base

**class** angr.analyses.decompiler.dephication.dephication_base.**DephicationBase**

Bases: *Analysis*

This is the base class for analyses that removes phi expressions from AIL graphs, SequenceNodes, and other types of AIL statement containers.

**__init__**(*func*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_map=None*)

**Parameters**

- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_map** (*VariableMap | None*)

### angr.analyses.decompiler.dephication.graph_dephication

class angr.analyses.decompiler.dephication.graph_dephication.**GraphDephication**

Bases: *DephicationBase*

GraphDephication removes phi expressions from an AIL graph, essentially transforms a partial-SSA form of AIL graph to a normal AIL graph.

**__init__**(*func*, *ail_graph*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_map=None*)

**Parameters**

- **func** (*Function* | str) – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **vvar_to_vvar_mapping** (*dict[int, int] | None*)
- **rewrite** (*bool*)
- **variable_map** (VariableMap | *None*)

### angr.analyses.decompiler.dephication.graph_rewriting

class angr.analyses.decompiler.dephication.graph_rewriting.**GraphRewritingAnalysis**

Bases: *ForwardAnalysis*[None, NodeType, object, object, object]

This analysis traverses the AIL graph and rewrites virtual variables accordingly.

### angr.analyses.decompiler.dephication.graph_vvar_mapping

class angr.analyses.decompiler.dephication.graph_vvar_mapping.**GraphDephicationVVarMapping**

Bases: *Analysis*

GraphDephicationVVarMapping collects information from an AIL graph regarding how virtual variables should be mapped to each other in order to remove phi variables. This mapping will later be used to transform a partial-SSA form of AIL graph to a conventional AIL graph.

This class largely implements "Translating Out of Static Single Assignment Form" by Sreedhar et al.

**__init__**(*func*, *ail_graph*, *ail_manager*, *entry=None*, *vvar_id_start=0*, *arg_vvars=None*)

**Parameters**

- **func** (*Function* | str) – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **ail_manager** (*Manager*)
- **vvar_id_start** (*int*)
- **arg_vvars** (*list[VirtualVariable] | None*)

### angr.analyses.decompiler.dephication.rewriting_engine

class angr.analyses.decompiler.dephication.rewriting_engine.**SimEngineDephiRewriting**

Bases: *SimEngineNostmtAIL*[None, *Expression* | None, *Statement* | tuple[*Statement*, ...], None]

This engine rewrites every block to insert phi variables and replaces every used variable with their versioned copies at each use location.

**state:** None

**__init__**(*project*, *vvar_to_vvar*, *func_addr=None*, *kb=None*, *variable_map=None*)

**Parameters**

- **vvar_to_vvar** (*dict[int, int]*)
- **func_addr** (*int | None*)
- **kb** (KnowledgeBase | *None*)
- **variable_map** (VariableMap | *None*)

**append_statement**(*stmt*)

**Return type**
None

**Parameters**
**stmt** (Statement)

### angr.analyses.decompiler.dephication.seqnode_dephication

**class** angr.analyses.decompiler.dephication.seqnode_dephication.**PhiAssignmentCollector**

Bases: *SequenceWalker*

PhiAssignmentCollector collects all phi variables and their corresponding source virtual variables in a SequenceNode.

**__init__**(*seq_node*)

**Parameters**
**seq_node** (SequenceNode)

**class** angr.analyses.decompiler.dephication.seqnode_dephication.**SeqNodeRewriter**

Bases: *SequenceWalker*

SeqNodeRewriter rewrites a SequenceNode by replacing all phi variables with their corresponding source virtual variables.

**__init__**(*seq_node*, *vvar_to_vvar*, *project*, *func_addr=None*, *kb=None*, *variable_map=None*)

**Parameters**

- **seq_node** (SequenceNode)
- **vvar_to_vvar** (*dict[int, int]*)
- **project** (Project)
- **func_addr** (*int | None*)
- **kb** (KnowledgeBase | *None*)
- **variable_map** (VariableMap | *None*)

**class** angr.analyses.decompiler.dephication.seqnode_dephication.**SeqNodeDephication**

Bases: *DephicationBase*

SeqNodeDephication removes phi expressions from a SequenceNode and its children. It also removes redundant variable assignments, e.g., *vvar_2 = vvar_1* where both vvar_1 and vvar_2 are mapped to the same variable.

__init__(*func*, *seq_node*, *vvar_to_vvar_mapping=None*, *rewrite=False*, *variable_map=None*)

**Parameters**

- **func** (Function | *str*)
- **vvar_to_vvar_mapping** (*dict[int, int]* | *None*)
- **rewrite** (*bool*)
- **variable_map** (VariableMap | *None*)

### angr.analyses.decompiler.dirty_rewriters

### Submodules

*amd64_dirty*
*rewriter_base*

### angr.analyses.decompiler.dirty_rewriters.amd64_dirty

**class** angr.analyses.decompiler.dirty_rewriters.amd64_dirty.**AMD64DirtyRewriter**

Bases: *DirtyRewriterBase*

Rewrites AMD64 DirtyStatement and DirtyExpression.

### angr.analyses.decompiler.dirty_rewriters.rewriter_base

**class** angr.analyses.decompiler.dirty_rewriters.rewriter_base.**DirtyRewriterBase**

Bases: object

The base class for DirtyStatement and DirtyExpression rewriters.

__init__(*dirty*, *arch*, *manager*)

**Parameters**

- **dirty** (DirtyExpression | DirtyStatement)
- **manager** (*Manager*)

**arch**

**manager**

**result:** *Expression* | *Statement* | None

### angr.analyses.decompiler.edits

Knowledge-base edits for decompilation output: renaming, retyping, and commenting.

This layer is deliberately free of any UI or transport dependency so both the headless MCP server and angr-management drive the same code. It is *not* thread-safe: locking belongs to whoever owns the knowledge base's lifetime.

**exception** angr.analyses.decompiler.edits.**AmbiguousFunctionError**

Bases: *DecompilationEditError*

More than one function matches the given name. Renames make this reachable.

**exception** angr.analyses.decompiler.edits.**DecompilationEditError**

Bases: *AngrError*

Base class for every failure raised by the decompilation edit layer.

**class** angr.analyses.decompiler.edits.**EditHooks**

Bases: Protocol

Notifications fired immediately *before* each mutation, while the old value is still readable from the knowledge base.

The method set mirrors angr-management's plugin hooks one-for-one so its adapter is a pure forwarder. Firing before the mutation matches what the GUI's own edit dialogs do, and is what lets a handler snapshot pre-edit state.

**before_function_renamed**(*func*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **old_name** (*str*)
- **new_name** (*str*)

**before_stack_var_renamed**(*func*, *offset*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_func_arg_renamed**(*func*, *arg_index*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_global_var_renamed**(*addr*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **addr** (*int*)

- **old_name** (*str*)
- **new_name** (*str*)

**before_stack_var_retyped**(*func*, *offset*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_func_arg_retyped**(*func*, *arg_index*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_global_var_retyped**(*addr*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_other_var_retyped**(*var*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **var** (SimVariable)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_function_retyped**(*func*, *old_proto*, *new_proto*)

**Return type**
None

**Parameters**

- **func** (Function)

- **old_proto** (SimTypeFunction | None)
- **new_proto** (SimTypeFunction)

**before_comment_changed**(*addr*, *old*, *new*, *created*, *decomp*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old** (*str*)
- **new** (*str*)
- **created** (*bool*)
- **decomp** (*bool*)

**__init__**(**args*, ***kwargs*)

class angr.analyses.decompiler.edits.**EditResult**

Bases: object

The outcome of a single edit. changed is False when the edit was a no-op.

**changed:** bool

**kind:** str

**func_addr:** int | None = None

**old:** Any = None

**new:** Any = None

**refresh:** *Refresh*

**detail:** dict[str, Any]

**__init__**(*changed*, *kind*, *func_addr=None*, *old=None*, *new=None*, *refresh=<factory>*, *detail=<factory>*)

**Parameters**

- **changed** (*bool*)
- **kind** (*str*)
- **func_addr** (*int | None*)
- **old** (*Any*)
- **new** (*Any*)
- **refresh** (Refresh)
- **detail** (*dict[str, Any]*)

**Return type**
None

**exception** angr.analyses.decompiler.edits.**FunctionNotFoundError**

Bases: *DecompilationEditError*

No function matches the given address or name.

**exception** angr.analyses.decompiler.edits.**InvalidNameError**

Bases: *DecompilationEditError*

The requested name is not a usable identifier.

**exception** angr.analyses.decompiler.edits.**NameCollisionError**

Bases: *DecompilationEditError*

The requested name is already bound to a different function, variable, or label.

**exception** angr.analyses.decompiler.edits.**NotDecompiledError**

Bases: *DecompilationEditError*

The function has no cached decompilation, so there is nothing to edit.

**class** angr.analyses.decompiler.edits.**NullEditHooks**

Bases: object

A concrete no-op implementation. Subclass it so an adapter only overrides what it needs.

**before_function_renamed**(*func*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **old_name** (*str*)
- **new_name** (*str*)

**before_stack_var_renamed**(*func*, *offset*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_func_arg_renamed**(*func*, *arg_index*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

before_global_var_renamed(*addr*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

before_stack_var_retyped(*func*, *offset*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

before_func_arg_retyped(*func*, *arg_index*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

before_global_var_retyped(*addr*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

before_other_var_retyped(*var*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **var** (SimVariable)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_function_retyped**(*func*, *old_proto*, *new_proto*)

**Return type**
None

**Parameters**

- **func** (Function)
- **old_proto** (SimTypeFunction | *None*)
- **new_proto** (SimTypeFunction)

**before_comment_changed**(*addr*, *old*, *new*, *created*, *decomp*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old** (*str*)
- **new** (*str*)
- **created** (*bool*)
- **decomp** (*bool*)

**class** angr.analyses.decompiler.edits.**Refresh**

Bases: object

What a caller has to redo after an edit.

Lets a UI pick between re-rendering text, rebuilding the codegen AST, and a full re-decompilation instead of guessing. `text_stale_all` means every cached decompilation's rendered text is stale (a function rename changes call sites everywhere), but the caller should re-render lazily: eagerly re-rendering the whole cache would thrash its LRU.

**text_stale:** frozenset[int] = frozenset({})

**text_stale_all:** bool = False

**reanalyze:** frozenset[int] = frozenset({})

**redecompile:** frozenset[int] = frozenset({})

**function_list_dirty:** bool = False

**disassembly_dirty:** bool = False

**merge**(*other*)

**Return type**
*Refresh*

**Parameters**
**other** (Refresh)

**__init__**(*text_stale=frozenset({})*, *text_stale_all=False*, *reanalyze=frozenset({})*, *redecompile=frozenset({})*, *function_list_dirty=False*, *disassembly_dirty=False*)

**Parameters**

- **text_stale** (*frozenset[int]*)
- **text_stale_all** (*bool*)
- **reanalyze** (*frozenset[int]*)
- **redecompile** (*frozenset[int]*)
- **function_list_dirty** (*bool*)
- **disassembly_dirty** (*bool*)

**Return type**
None

**class** angr.analyses.decompiler.edits.**ResolvedVariable**

Bases: object

A pseudocode display name resolved to the objects an edit needs.

`variable` is the concrete/SSA variable that `set_variable_type` expects; `unified` is the unified variable that a rename mutates (None for globals, which have no unified form).

**kind:** Literal['argument', 'local', 'global']

**variable:** *SimVariable*

**unified:** *SimVariable* | None

**cvar:** *CVariable* | None = None

**arg_index:** int | None = None

**stack_offset:** int | None = None

**global_addr:** int | None = None

**ambiguous:** bool = False

**property name:** str | None

**property storage:** str

**detail**()

**Return type**
dict

**__init__**(*kind*, *variable*, *unified*, *cvar=None*, *arg_index=None*, *stack_offset=None*, *global_addr=None*, *ambiguous=False*)

**Parameters**

- **kind** (*Literal['argument', 'local', 'global']*)
- **variable** (SimVariable)
- **unified** (SimVariable | *None*)
- **cvar** (CVariable | *None*)
- **arg_index** (*int* | *None*)
- **stack_offset** (*int* | *None*)
- **global_addr** (*int* | *None*)

- **ambiguous** (*bool*)

**Return type**
None

exception angr.analyses.decompiler.edits.**TypeParseError**

Bases: *DecompilationEditError*

A C type declaration or function signature could not be parsed.

exception angr.analyses.decompiler.edits.**UnsupportedEditError**

Bases: *DecompilationEditError*

The requested edit is not supported for this kind of target.

exception angr.analyses.decompiler.edits.**VariableNotFoundError**

Bases: *DecompilationEditError*

No variable in the decompilation matches the given display name.

`candidates` carries the names that *are* available, so a caller that failed can correct itself without a second round trip.

angr.analyses.decompiler.edits.**concrete_variables**(*varman*, *resolved*)

Every SSA variable sharing the resolved variable's unified form.

Retyping applies to all of them. Computed here rather than relying on `set_variable_type(all_unified=True)`, which silently does nothing when the variable it is handed is not a key in the SSA-to-unified map.

**Return type**
`list`[*SimVariable*]

**Parameters**
**resolved** (`ResolvedVariable`)

angr.analyses.decompiler.edits.**get_cache**(*kb*, *func_addr*, *flavor='pseudocode'*)

Return the cached decompilation for a function, or None if there is none.

**Return type**
*DecompilationCache* | `None`

**Parameters**

- **kb** (`KnowledgeBase`)
- **func_addr** (*int*)
- **flavor** (*str*)

angr.analyses.decompiler.edits.**global_variable_at**(*kb*, *addr*)

The global SimVariable recorded at an address, if variable recovery produced one.

**Parameters**

- **kb** (`KnowledgeBase`)
- **addr** (*int*)

angr.analyses.decompiler.edits.**invalidate**(*kb*, *func_addr*, ***, *flavors=None*, *drop_variables=False*)

Drop cached decompilations for a function.

**Parameters**

- **flavors** (Iterable[str] | None) – Flavors to drop; None drops every flavor currently cached. A prototype change must drop all of them, or a non-pseudocode flavor silently retains the old signature.
- **drop_variables** (bool) – Also drop `kb.dec_variables[func_addr]`. Required for new argument names to take effect, but it discards every rename and manual type for the function – see `snapshot_user_edits()`.
- **kb** (KnowledgeBase)
- **func_addr** (*int*)

**Return type**
None

angr.analyses.decompiler.edits.**list_variable_names**(*codegen*, *kb=None*, *func_addr=None*)

Every display name addressable in this decompilation, for error messages.

**Return type**
list[str]

**Parameters**

- **kb** (KnowledgeBase | *None*)
- **func_addr** (*int* | *None*)

angr.analyses.decompiler.edits.**parse_address**(*value*)

Parse an address given as an int or a string. Accepts 0x-prefixed hex and decimal.

**Return type**
int

**Parameters**
**value** (*str* | *int*)

angr.analyses.decompiler.edits.**reflow_types**(*project*, *func*, ***, *kb=None*, *flavor='pseudocode'*, *rerender=True*)

Re-run type inference over the cached constraints and refresh the rendered code.

Separate from `set_variable_type()` so a batch can retype many variables and reflow once; per-variable reflow would re-run Typehoon N times.

This is not a re-decompilation: the AST is untouched, so a retype that should change how an access renders (a struct field, an array index) needs a full re-decompilation instead.

**Parameters**

- **project** (Project)
- **func** (Function)
- **kb** (KnowledgeBase | *None*)
- **flavor** (*str*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.**rename_function**(*project*, *func*, *new_name*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *allow_overwrite=True*, *strict_names=True*, *rerender=True*)

Rename a function.

Nothing is invalidated: other functions' cached ASTs reference the same Function object, so only their rendered text goes stale. That is reported through `Refresh.text_stale_all` for the caller to act on lazily.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **new_name** (*str*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **allow_overwrite** (*bool*)
- **strict_names** (*bool*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.**rename_global**(*project*, *addr*, *new_name*, ***, *kb=None*, *hooks=None*, *allow_overwrite=True*, *strict_names=True*)

Rename a global by address, without needing a function whose decompilation shows it.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **addr** (*int*)
- **new_name** (*str*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **allow_overwrite** (*bool*)
- **strict_names** (*bool*)

angr.analyses.decompiler.edits.**rename_variable**(*project*, *func*, *variable_name*, *new_name*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *allow_overwrite=True*, *strict_names=True*, *rerender=True*, *codegen=None*)

Rename a local, an argument, or a global as it appears in a function's decompilation.

Sets `renamed` on the variable, without which a later re-decompilation overwrites the name.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **variable_name** (*str*)

- **new_name** (*str*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **allow_overwrite** (*bool*)
- **strict_names** (*bool*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.**require_cache**(*kb*, *func_addr*, *flavor='pseudocode'*)

Return the cached decompilation, raising NotDecompiledError if it is missing or empty.

**Return type**
*DecompilationCache*

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)
- **flavor** (*str*)

angr.analyses.decompiler.edits.**resolve_function**(*kb*, ***, *address=None*, *name=None*, *containing=True*)

Find a function by address or by name.

**Parameters**

- **containing** (bool) – If True, an address inside a function resolves to that function rather than requiring the exact entry address.
- **kb** (KnowledgeBase)
- **address** (*str* | *int* | *None*)
- **name** (*str* | *None*)

**Return type**
*Function*

angr.analyses.decompiler.edits.**resolve_variable**(*kb*, *func_addr*, *display_name*, ***, *codegen=None*, *flavor='pseudocode'*)

Resolve a name as it appears in the pseudocode to the underlying variable.

Precedence is argument > local > global. A name matching more than one variable within the same bucket resolves deterministically and sets `ambiguous`, rather than failing – a batch edit should be able to report the collision and continue.

**Return type**
*ResolvedVariable*

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)
- **display_name** (*str*)
- **flavor** (*str*)

angr.analyses.decompiler.edits.**restore_user_edits**(*kb*, *func_addr*, *snapshot*)

Re-apply a `snapshot_user_edits()` result after re-decompilation.

Best-effort: idents can change across a re-decompile, so the unmatched ones are returned rather than silently dropped.

**Return type**
tuple[int, list[str]]

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)
- **snapshot** (*dict[str, tuple[str | None, SimType | None]]*)

angr.analyses.decompiler.edits.**set_comment**(*project*, *addr*, *comment*, ***, *kind=None*, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *mirror_to_pseudocode=True*, *snap=True*, *rerender=True*)

Set the comment at an address, or clear it with an empty string or None.

The comment goes into kb.comments, which the disassembly renders and which the decompiler reads for the function header. Per-statement pseudocode comments live in codegen.stmt_comments instead, so both are written – except at the function entry, where kb.comments is already what the header renders and mirroring would show the comment twice.

`kind` sets the comment's display kind; None keeps whatever kind the comment already has. Clearing a comment always clears its kind.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **addr** (*int*)
- **comment** (*str | None*)
- **kind** (CommentKind | *None*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **mirror_to_pseudocode** (*bool*)
- **snap** (*bool*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.**set_function_prototype**(*project*, *func*, *prototype*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *invalidate_cache=True*, *preserve_user_edits=True*, *redecompile=False*)

Set a function's prototype.

The function name inside the signature is ignored; use `rename_function()` to rename.

Dropping kb.dec_variables is what makes new argument names take effect, but it also discards every rename and manual type for the function. Those are snapshotted and, when this function re-decompiles, restored. Otherwise the snapshot is returned in `detail["user_edits"]` so a caller that decompiles asynchronously can restore it once its own job finishes.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **prototype** (*str* | SimTypeFunction)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **invalidate_cache** (*bool*)
- **preserve_user_edits** (*bool*)
- **redecompile** (*bool*)

angr.analyses.decompiler.edits.**set_global_type**(*project*, *addr*, *c_type*, ***, *kb=None*, *hooks=None*)

Set the type of a global by address.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **addr** (*int*)
- **c_type** (*str* | SimType)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)

angr.analyses.decompiler.edits.**set_variable_type**(*project*, *func*, *variable_name*, *c_type*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *reflow=True*, *rerender=True*, *allow_prototype_change=True*, *codegen=None*)

Change the type of a local, an argument, or a global.

Arguments are retyped by rewriting the function prototype, which requires a re-decompilation – the returned Refresh says so. Pass `allow_prototype_change=False` to refuse instead.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **variable_name** (*str*)
- **c_type** (*str* | SimType)

- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **reflow** (*bool*)
- **rerender** (*bool*)
- **allow_prototype_change** (*bool*)

angr.analyses.decompiler.edits.**snapshot_user_edits**(*kb*, *func_addr*)

Capture user renames and manual types for a function, keyed by `SimVariable.ident`.

Used to survive the `dec_variables` drop that a prototype change requires.

**Return type**
dict[str, tuple[str | None, *SimType* | None]]

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)

angr.analyses.decompiler.edits.**validate_name**(*name*, *, *strict=True*)

Reject names that cannot be used as identifiers.

`strict` requires a C identifier. Without it only whitespace-free names are required, which still admits things like `int;` that render as uncompilable C – so strict is the default.

**Return type**
None

**Parameters**

- **name** (*str*)
- **strict** (*bool*)

### Submodules

| | |
|---|---|
| *cache* | Decompilation-cache access for the edit layer. |
| *errors* | |
| *hooks* | |
| *ops* | The mutating operations of the edit layer. |
| *resolve* | Resolution helpers shared by every edit operation: address-or-name to Function, and pseudocode display name to the underlying SimVariable. |
| *results* | |

### angr.analyses.decompiler.edits.cache

Decompilation-cache access for the edit layer.

Both `kb.decompilations` and `kb.dec_variables` spill to LMDB under memory pressure and hand back a freshly deserialized – that is, *different* – Python object on reload. Never hold a `DecompilationCache` or a `VariableManagerInternal` across anything that can decompile; always re-fetch by key at the point of mutation.

angr.analyses.decompiler.edits.cache.**get_cache**(*kb*, *func_addr*, *flavor='pseudocode'*)

Return the cached decompilation for a function, or None if there is none.

**Return type**
*DecompilationCache* | None

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)
- **flavor** (*str*)

angr.analyses.decompiler.edits.cache.**require_cache**(*kb*, *func_addr*, *flavor='pseudocode'*)

Return the cached decompilation, raising NotDecompiledError if it is missing or empty.

**Return type**
*DecompilationCache*

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)
- **flavor** (*str*)

angr.analyses.decompiler.edits.cache.**invalidate**(*kb*, *func_addr*, ***, *flavors=None*, *drop_variables=False*)

Drop cached decompilations for a function.

**Parameters**

- **flavors** (Iterable[str] | None) – Flavors to drop; None drops every flavor currently cached. A prototype change must drop all of them, or a non-pseudocode flavor silently retains the old signature.
- **drop_variables** (bool) – Also drop `kb.dec_variables[func_addr]`. Required for new argument names to take effect, but it discards every rename and manual type for the function – see *snapshot_user_edits()*.
- **kb** (KnowledgeBase)
- **func_addr** (*int*)

**Return type**
None

angr.analyses.decompiler.edits.cache.**snapshot_user_edits**(*kb*, *func_addr*)

Capture user renames and manual types for a function, keyed by `SimVariable.ident`.

Used to survive the `dec_variables` drop that a prototype change requires.

**Return type**
dict[str, tuple[str | None, *SimType* | None]]

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)

angr.analyses.decompiler.edits.cache.**restore_user_edits**(*kb*, *func_addr*, *snapshot*)

Re-apply a *snapshot_user_edits()* result after re-decompilation.

Best-effort: idents can change across a re-decompile, so the unmatched ones are returned rather than silently dropped.

**Return type**
tuple[int, list[str]]

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)
- **snapshot** (*dict[str, tuple[str | None, SimType | None]]*)

angr.analyses.decompiler.edits.cache.**function_summary**(*kb*, *func_addr*)

Small helper for edit results: what is currently cached for a function.

**Return type**
dict[str, Any]

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)

### angr.analyses.decompiler.edits.errors

**exception** angr.analyses.decompiler.edits.errors.**DecompilationEditError**

Bases: *AngrError*

Base class for every failure raised by the decompilation edit layer.

**exception** angr.analyses.decompiler.edits.errors.**FunctionNotFoundError**

Bases: *DecompilationEditError*

No function matches the given address or name.

**exception** angr.analyses.decompiler.edits.errors.**AmbiguousFunctionError**

Bases: *DecompilationEditError*

More than one function matches the given name. Renames make this reachable.

**exception** angr.analyses.decompiler.edits.errors.**VariableNotFoundError**

Bases: *DecompilationEditError*

No variable in the decompilation matches the given display name.

`candidates` carries the names that *are* available, so a caller that failed can correct itself without a second round trip.

**exception** angr.analyses.decompiler.edits.errors.**NotDecompiledError**

Bases: *DecompilationEditError*

The function has no cached decompilation, so there is nothing to edit.

**exception** angr.analyses.decompiler.edits.errors.**InvalidNameError**

Bases: *DecompilationEditError*

The requested name is not a usable identifier.

**exception** angr.analyses.decompiler.edits.errors.**NameCollisionError**

Bases: *DecompilationEditError*

The requested name is already bound to a different function, variable, or label.

**exception** angr.analyses.decompiler.edits.errors.**TypeParseError**

Bases: *DecompilationEditError*

A C type declaration or function signature could not be parsed.

**exception** angr.analyses.decompiler.edits.errors.**UnsupportedEditError**

Bases: *DecompilationEditError*

The requested edit is not supported for this kind of target.

### angr.analyses.decompiler.edits.hooks

**class** angr.analyses.decompiler.edits.hooks.**EditHooks**

Bases: Protocol

Notifications fired immediately *before* each mutation, while the old value is still readable from the knowledge base.

The method set mirrors angr-management's plugin hooks one-for-one so its adapter is a pure forwarder. Firing before the mutation matches what the GUI's own edit dialogs do, and is what lets a handler snapshot pre-edit state.

**before_function_renamed**(*func*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **old_name** (*str*)
- **new_name** (*str*)

**before_stack_var_renamed**(*func*, *offset*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_func_arg_renamed**(*func*, *arg_index*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)

- **old_name** (*str*)
- **new_name** (*str*)

**before_global_var_renamed**(*addr*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_stack_var_retyped**(*func*, *offset*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_func_arg_retyped**(*func*, *arg_index*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_global_var_retyped**(*addr*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_other_var_retyped**(*var*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **var** (SimVariable)

- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_function_retyped**(*func*, *old_proto*, *new_proto*)

**Return type**
None

**Parameters**

- **func** (Function)
- **old_proto** (SimTypeFunction | *None*)
- **new_proto** (SimTypeFunction)

**before_comment_changed**(*addr*, *old*, *new*, *created*, *decomp*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old** (*str*)
- **new** (*str*)
- **created** (*bool*)
- **decomp** (*bool*)

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.edits.hooks.**NullEditHooks**

Bases: object

A concrete no-op implementation. Subclass it so an adapter only overrides what it needs.

**before_function_renamed**(*func*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **old_name** (*str*)
- **new_name** (*str*)

**before_stack_var_renamed**(*func*, *offset*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_func_arg_renamed**(*func*, *arg_index*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_global_var_renamed**(*addr*, *old_name*, *new_name*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old_name** (*str*)
- **new_name** (*str*)

**before_stack_var_retyped**(*func*, *offset*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **offset** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_func_arg_retyped**(*func*, *arg_index*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **func** (Function)
- **arg_index** (*int*)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_global_var_retyped**(*addr*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old_type** (SimType | *None*)

- **new_type** (SimType)

**before_other_var_retyped**(*var*, *old_type*, *new_type*)

**Return type**
None

**Parameters**

- **var** (SimVariable)
- **old_type** (SimType | *None*)
- **new_type** (SimType)

**before_function_retyped**(*func*, *old_proto*, *new_proto*)

**Return type**
None

**Parameters**

- **func** (Function)
- **old_proto** (SimTypeFunction | *None*)
- **new_proto** (SimTypeFunction)

**before_comment_changed**(*addr*, *old*, *new*, *created*, *decomp*)

**Return type**
None

**Parameters**

- **addr** (*int*)
- **old** (*str*)
- **new** (*str*)
- **created** (*bool*)
- **decomp** (*bool*)

angr.analyses.decompiler.edits.hooks.**coerce_hooks**(*hooks*)

**Return type**
*EditHooks*

**Parameters**
**hooks** (EditHooks | *None*)

### angr.analyses.decompiler.edits.ops

The mutating operations of the edit layer.

Every operation re-fetches the decompilation cache by key at the point of mutation rather than holding a reference: see the note in *cache*.

angr.analyses.decompiler.edits.ops.**rename_function**(*project*, *func*, *new_name*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *allow_overwrite=True*, *strict_names=True*, *rerender=True*)

Rename a function.

Nothing is invalidated: other functions' cached ASTs reference the same Function object, so only their rendered text goes stale. That is reported through `Refresh.text_stale_all` for the caller to act on lazily.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **new_name** (*str*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **allow_overwrite** (*bool*)
- **strict_names** (*bool*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.ops.**rename_variable**(*project*, *func*, *variable_name*, *new_name*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *allow_overwrite=True*, *strict_names=True*, *rerender=True*, *codegen=None*)

Rename a local, an argument, or a global as it appears in a function's decompilation.

Sets `renamed` on the variable, without which a later re-decompilation overwrites the name.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **variable_name** (*str*)
- **new_name** (*str*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **allow_overwrite** (*bool*)
- **strict_names** (*bool*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.ops.**reflow_types**(*project*, *func*, ***, *kb=None*, *flavor='pseudocode'*, *rerender=True*)

Re-run type inference over the cached constraints and refresh the rendered code.

Separate from *set_variable_type()* so a batch can retype many variables and reflow once; per-variable reflow would re-run Typehoon N times.

This is not a re-decompilation: the AST is untouched, so a retype that should change how an access renders (a struct field, an array index) needs a full re-decompilation instead.

**Parameters**

- **project** (Project)
- **func** (Function)
- **kb** (KnowledgeBase | *None*)
- **flavor** (*str*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.ops.**set_variable_type**(*project*, *func*, *variable_name*, *c_type*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *reflow=True*, *rerender=True*, *allow_prototype_change=True*, *codegen=None*)

Change the type of a local, an argument, or a global.

Arguments are retyped by rewriting the function prototype, which requires a re-decompilation – the returned Refresh says so. Pass `allow_prototype_change=False` to refuse instead.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **variable_name** (*str*)
- **c_type** (*str* | SimType)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **reflow** (*bool*)
- **rerender** (*bool*)
- **allow_prototype_change** (*bool*)

angr.analyses.decompiler.edits.ops.**set_function_prototype**(*project*, *func*, *prototype*, ***, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *invalidate_cache=True*, *preserve_user_edits=True*, *redecompile=False*)

Set a function's prototype.

The function name inside the signature is ignored; use *rename_function()* to rename.

Dropping kb.dec_variables is what makes new argument names take effect, but it also discards every rename and manual type for the function. Those are snapshotted and, when this function re-decompiles, restored. Otherwise

the snapshot is returned in `detail["user_edits"]` so a caller that decompiles asynchronously can restore it once its own job finishes.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **func** (Function)
- **prototype** (*str* | SimTypeFunction)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **invalidate_cache** (*bool*)
- **preserve_user_edits** (*bool*)
- **redecompile** (*bool*)

angr.analyses.decompiler.edits.ops.**set_comment**(*project*, *addr*, *comment*, *, *kind=None*, *kb=None*, *hooks=None*, *flavor='pseudocode'*, *mirror_to_pseudocode=True*, *snap=True*, *rerender=True*)

Set the comment at an address, or clear it with an empty string or None.

The comment goes into kb.comments, which the disassembly renders and which the decompiler reads for the function header. Per-statement pseudocode comments live in codegen.stmt_comments instead, so both are written – except at the function entry, where kb.comments is already what the header renders and mirroring would show the comment twice.

`kind` sets the comment's display kind; None keeps whatever kind the comment already has. Clearing a comment always clears its kind.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **addr** (*int*)
- **comment** (*str* | *None*)
- **kind** (CommentKind | *None*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **flavor** (*str*)
- **mirror_to_pseudocode** (*bool*)
- **snap** (*bool*)
- **rerender** (*bool*)

angr.analyses.decompiler.edits.ops.**global_variable_at**(*kb*, *addr*)

The global SimVariable recorded at an address, if variable recovery produced one.

**Parameters**

- **kb** (KnowledgeBase)
- **addr** (*int*)

angr.analyses.decompiler.edits.ops.**rename_global**(*project*, *addr*, *new_name*, *, *kb=None*, *hooks=None*, *allow_overwrite=True*, *strict_names=True*)

Rename a global by address, without needing a function whose decompilation shows it.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **addr** (*int*)
- **new_name** (*str*)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)
- **allow_overwrite** (*bool*)
- **strict_names** (*bool*)

angr.analyses.decompiler.edits.ops.**set_global_type**(*project*, *addr*, *c_type*, *, *kb=None*, *hooks=None*)

Set the type of a global by address.

**Return type**
*EditResult*

**Parameters**

- **project** (Project)
- **addr** (*int*)
- **c_type** (*str* | SimType)
- **kb** (KnowledgeBase | *None*)
- **hooks** (EditHooks | *None*)

### angr.analyses.decompiler.edits.resolve

Resolution helpers shared by every edit operation: address-or-name to Function, and pseudocode display name to the underlying SimVariable.

angr.analyses.decompiler.edits.resolve.**parse_address**(*value*)

Parse an address given as an int or a string. Accepts 0x-prefixed hex and decimal.

**Return type**
int

**Parameters**
**value** (*str* | *int*)

angr.analyses.decompiler.edits.resolve.**validate_name**(*name*, *, *strict=True*)

Reject names that cannot be used as identifiers.

`strict` requires a C identifier. Without it only whitespace-free names are required, which still admits things like `int;` that render as uncompilable C – so strict is the default.

**Return type**
None

**Parameters**

- **name** (*str*)
- **strict** (*bool*)

angr.analyses.decompiler.edits.resolve.**resolve_function**(*kb*, *, *address=None*, *name=None*, *containing=True*)

Find a function by address or by name.

**Parameters**

- **containing** (bool) – If True, an address inside a function resolves to that function rather than requiring the exact entry address.
- **kb** (KnowledgeBase)
- **address** (*str | int | None*)
- **name** (*str | None*)

**Return type**
*Function*

**class** angr.analyses.decompiler.edits.resolve.**ResolvedVariable**

Bases: object

A pseudocode display name resolved to the objects an edit needs.

`variable` is the concrete/SSA variable that `set_variable_type` expects; `unified` is the unified variable that a rename mutates (None for globals, which have no unified form).

**kind:** Literal['argument', 'local', 'global']

**variable:** *SimVariable*

**unified:** *SimVariable* | None

**cvar:** *CVariable* | None = None

**arg_index:** int | None = None

**stack_offset:** int | None = None

**global_addr:** int | None = None

**ambiguous:** bool = False

**property name:** str | None

**property storage:** str

**detail**()

**Return type**
dict

**__init__**(*kind*, *variable*, *unified*, *cvar=None*, *arg_index=None*, *stack_offset=None*, *global_addr=None*, *ambiguous=False*)

**Parameters**

- **kind** (*Literal['argument', 'local', 'global']*)
- **variable** (SimVariable)
- **unified** (SimVariable | *None*)
- **cvar** (CVariable | *None*)
- **arg_index** (*int* | *None*)
- **stack_offset** (*int* | *None*)
- **global_addr** (*int* | *None*)
- **ambiguous** (*bool*)

**Return type**
None

angr.analyses.decompiler.edits.resolve.**concrete_variables**(*varman*, *resolved*)

Every SSA variable sharing the resolved variable's unified form.

Retyping applies to all of them. Computed here rather than relying on `set_variable_type(all_unified=True)`, which silently does nothing when the variable it is handed is not a key in the SSA-to-unified map.

**Return type**
list[*SimVariable*]

**Parameters**
**resolved** (ResolvedVariable)

angr.analyses.decompiler.edits.resolve.**list_variable_names**(*codegen*, *kb=None*, *func_addr=None*)

Every display name addressable in this decompilation, for error messages.

**Return type**
list[str]

**Parameters**

- **kb** (KnowledgeBase | *None*)
- **func_addr** (*int* | *None*)

angr.analyses.decompiler.edits.resolve.**resolve_variable**(*kb*, *func_addr*, *display_name*, ***, *codegen=None*, *flavor='pseudocode'*)

Resolve a name as it appears in the pseudocode to the underlying variable.

Precedence is argument > local > global. A name matching more than one variable within the same bucket resolves deterministically and sets `ambiguous`, rather than failing – a batch edit should be able to report the collision and continue.

**Return type**
*ResolvedVariable*

**Parameters**

- **kb** (KnowledgeBase)
- **func_addr** (*int*)
- **display_name** (*str*)
- **flavor** (*str*)

### angr.analyses.decompiler.edits.results

**class** angr.analyses.decompiler.edits.results.**Refresh**

Bases: object

What a caller has to redo after an edit.

Lets a UI pick between re-rendering text, rebuilding the codegen AST, and a full re-decompilation instead of guessing. `text_stale_all` means every cached decompilation's rendered text is stale (a function rename changes call sites everywhere), but the caller should re-render lazily: eagerly re-rendering the whole cache would thrash its LRU.

**text_stale:** frozenset[int] = frozenset({})

**text_stale_all:** bool = False

**reanalyze:** frozenset[int] = frozenset({})

**redecompile:** frozenset[int] = frozenset({})

**function_list_dirty:** bool = False

**disassembly_dirty:** bool = False

**merge**(*other*)

**Return type**
*Refresh*

**Parameters**
**other** (Refresh)

**__init__**(*text_stale=frozenset({})*, *text_stale_all=False*, *reanalyze=frozenset({})*, *redecompile=frozenset({})*, *function_list_dirty=False*, *disassembly_dirty=False*)

**Parameters**

- **text_stale** (*frozenset[int]*)
- **text_stale_all** (*bool*)
- **reanalyze** (*frozenset[int]*)
- **redecompile** (*frozenset[int]*)
- **function_list_dirty** (*bool*)
- **disassembly_dirty** (*bool*)

**Return type**
None

class angr.analyses.decompiler.edits.results.**EditResult**

Bases: object

The outcome of a single edit. changed is False when the edit was a no-op.

**changed:** bool

**kind:** str

**func_addr:** int | None = None

**old:** Any = None

**new:** Any = None

**refresh:** *Refresh*

**detail:** dict[str, Any]

**__init__**(*changed*, *kind*, *func_addr=None*, *old=None*, *new=None*, *refresh=<factory>*, *detail=<factory>*)

**Parameters**

- **changed** (*bool*)
- **kind** (*str*)
- **func_addr** (*int | None*)
- **old** (*Any*)
- **new** (*Any*)
- **refresh** (Refresh)
- **detail** (*dict[str, Any]*)

**Return type**
None

### angr.analyses.decompiler.empty_node_remover

class angr.analyses.decompiler.empty_node_remover.**EmptyNodeRemover**

Bases: object

Rewrites a node and its children to remove empty nodes.

The following optimizations are performed at the same time: - Convert if (A) { } else { ... } to if(!A) { ... } else { }

**Variables**
**_claripy_ast_conditions** – True if all node conditions are claripy ASTs. False if all node conditions are AIL expressions.

**__init__**(*node*, *ail_manager*, *claripy_ast_conditions=True*)

**Parameters**

- **ail_manager** (*Manager*)
- **claripy_ast_conditions** (*bool*)

### angr.analyses.decompiler.expression_narrower

**class** angr.analyses.decompiler.expression_narrower.**ExprNarrowingInfo**

Bases: object

Stores the analysis result of _narrowing_needed().

**__init__**(*narrowable*, *to_size=None*, *use_exprs=None*, *phi_vars=None*)

**Parameters**

- **narrowable** (*bool*)
- **to_size** (*int | None*)
- **use_exprs** (*list[tuple[VirtualVariable, AILCodeLocation]] | None*)
- **phi_vars** (*set[VirtualVariable] | None*)

**narrowable**

**to_size**

**use_exprs**

**phi_vars**

**class** angr.analyses.decompiler.expression_narrower.**EffectiveSizeExtractor**

Bases: *AILBlockWalker*[None, None, None]

Walks a statement once and extracts the effective size (in bits) of every virtual variable that appears in it.

For example, for virtual variable rax, *(rax & 0xff) + 0x1* means the effective size of rax is 8 bits, from bit 0 to bit 7. We record this information in vvar_effective_bits as {rax.varid: {rax.idx: (0, 8)}}.

We pay special consideration to expressions that are used as Call arguments, as they may have been converted to a smaller size because the Call argument needs that size, but the Call prototype may have been incorrectly inferred.

A single walk records information for all virtual variables in the statement, so one walker instance can be queried for many different variables without re-walking the statement. Constraints that parent expressions impose on their children are tracked per expression node (keyed by the node's `idx`) during the walk; results are aggregated per (varid, expression idx) so that repeated occurrences of the same variable are kept separate.

**__init__**(*ignore_call_args=True*)

**Parameters**

**ignore_call_args** (*bool*)

**vvar_effective_bits: dict[int, dict[int, tuple[int, int]]]**

**vvar_call_arg_effective_bits: dict[int, tuple[int, int]]**

**vvars_used_as_insert_base: set[int]**

**class** angr.analyses.decompiler.expression_narrower.**ExpressionNarrower**

Bases: *AILBlockRewriter*

Narrows an expression regardless of whether the expression is a definition or a use.

__init__(*project*, *rd*, *manager*, *narrowables*, *addr2blocks*, *new_blocks*)

**Parameters**

- **manager** (*Manager*)
- **addr2blocks** (*dict[tuple[int, int | None], Block]*)
- **new_blocks** (*dict[Block, Block]*)

new_vvar_sizes: dict[int, int]

replacement_core_vvars: dict[int, list[VirtualVariable]]

walk(*block*)

**Parameters**
**block** (*Block*)

### angr.analyses.decompiler.goto_manager

class angr.analyses.decompiler.goto_manager.**Goto**

Bases: object

Describe the existence of a goto (jump) statement. May have multiple gotos with the same address (targets will differ).

__init__(*src_addr*, *dst_addr*, *src_idx=None*, *dst_idx=None*, *src_ins_addr=None*)

class angr.analyses.decompiler.goto_manager.**GotoManager**

Bases: object

Container class for all Gotos found in a function after decompilation structuring. This should be populated using GotoSimplifier.

__init__(*func*, *gotos=None*)

gotos_in_block(*block*)

**Return type**
set[*Goto*]

**Parameters**
**block** (*Block*)

is_goto_edge(*src*, *dst*)

**Parameters**

- **src** (*Block*)
- **dst** (*Block*)

find_goto_edges(*graph*)

This function finds all edges that are _potential_ gotos in the graph. The gotos are not guaranteed to be correct, but they are an approximation based on how the Phoenix structuring algorithm will select edges from the graph to be gotos in structuring.

**Return type**
list[tuple[Block, Block]]

**Parameters**
**graph** (*DiGraph*)

### angr.analyses.decompiler.jump_target_collector

**class** angr.analyses.decompiler.jump_target_collector.**JumpTargetCollector**

Bases: object

Collect all jump targets.

**__init__**(*node*)

### angr.analyses.decompiler.jumptable_entry_condition_rewriter

**class** angr.analyses.decompiler.jumptable_entry_condition_rewriter.**JumpTableEntryConditionRewriter**

Bases: *SequenceWalker*

Remove artificial jump table entry conditions that ConditionProcessor introduced when dealing with jump tables.

**__init__**(*jumptable_entry_conds*)

### angr.analyses.decompiler.known_patterns

Known code patterns: context-aware, declarative descriptions of library/macro idioms that compilers inline, plus machinery (KnownPatternFinder) to find them in Clinic AIL graphs and outline them back into calls.

Each idiom is defined once as a `KnownPatternTemplate` that instantiates itself into a concrete `KnownPattern` for the target's `PatternContext` (architecture / platform / C++ runtime), so a single definition covers 32- and 64-bit, libstdc++ and MSVC, etc.

**class** angr.analyses.decompiler.known_patterns.**PITE**

Bases: *PatternExpr*

Matches an ITE (ternary) expression `cond ? iftrue : iffalse`.

Needed because ITERegionConverter (AFTER_GLOBAL_SIMPLIFICATION) collapses `if (c) x = a; else x = b;` diamonds into a single ITE assignment before the KnownPatternOutliner runs, so a two-armed value-select idiom reaches the outliner as an expression, not as a PGraphPat region. (A triangle, with one arm empty as in the MSVC std::string::c_str SSO select, is not converted, which is why that idiom is still a PGraphPat.)

**cond:** *PatternExpr*

**iftrue:** *PatternExpr*

**iffalse:** *PatternExpr*

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*cond*, *iftrue*, *iffalse*, *name=None*)

**Parameters**

- **cond** (PatternExpr)
- **iftrue** (PatternExpr)
- **iffalse** (PatternExpr)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**AllOf**

Bases: *PatternGate*

Opens when every sub-gate opens.

**gates:** tuple[*PatternGate*, ...]

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence:** bool

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label:** str

**__init__**(*gates=<factory>*)

**Parameters**
**gates** (*tuple[PatternGate, ...]*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**AnyOf**

Bases: *PatternGate*

Opens when any sub-gate opens.

**gates:** tuple[*PatternGate*, ...]

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence:** bool

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label:** str

**__init__**(*gates=<factory>*)

**Parameters**
**gates** (*tuple[PatternGate, ...]*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**CorroboratedBy**

Bases: *PatternGate*

Opens in a function where at least one of `witnesses` already matched.

Witnesses are given as template ``name``s or ``call_name``s; the finder puts both spellings of every established pattern into the evidence set.

**witnesses:** frozenset[str]

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence:** bool

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label:** str

**__init__**(*witnesses*)

**Parameters**
**witnesses** (*frozenset[str]*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**CppRef**

Bases: object

A reference to a C++ class type registered in the `cpp::std` SimTypeCollection, keyed by its unique (mangled-style) name.

**unique_name:** str

**ptr:** bool = True

**resolve**(*arch*)

**Return type**
*SimType* | None

**Parameters**
**arch** (*Arch*)

__init__(*unique_name*, *ptr=True*)

**Parameters**

- **unique_name** (*str*)
- **ptr** (*bool*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**GateContext**

Bases: object

Everything a gate may look at.

**Variables**

- **ctx** – The target's *PatternContext* (arch, platform, C++ runtime, and the binary-evidence facts).
- **project** – The Project, for gates that need to look past the context.
- **evidence** – Names and call names of the patterns established in the function currently being matched: both the patterns that matched in the first stage and the known-pattern calls already present in the graph (outlined by an earlier round). Empty when gates are evaluated outside a function.

**ctx:** *PatternContext*

**project:** *Project* | None = None

**evidence:** frozenset[str] = frozenset({})

**with_evidence**(*evidence*)

**Return type**
*GateContext*

**Parameters**
**evidence** (*Iterable[str]*)

__init__(*ctx*, *project=None*, *evidence=frozenset({})*)

**Parameters**

- **ctx** (PatternContext)
- **project** (Project | *None*)
- **evidence** (*frozenset[str]*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**KnownPattern**

Bases: object

A known code idiom, described declaratively.

**Variables**

- **name** – Identifier, e.g. "std_string_length".
- **display_name** – Human-readable name, e.g. "std::string::length".

- **call_name** – Target name of the synthesized Call.
- **pattern** – The structural pattern to match.
- **params** – Ordered parameters; their captures become the call arguments, in order.
- **returnty** – Return type of the synthesized call.
- **returnty_factory** – Optional callable building a call-site-specific return type from the values of the constant `extra_args` captures (e.g. CONTAINING_RECORD returns a pointer to a record whose layout depends on the matched field offset). Falls back to `returnty` when it returns None or when the constant values are unavailable.
- **extra_args** – Names of (typically constant) captures appended to the call arguments after `params`.
- **arches** – Allowed `archinfo.Arch.name` values; None = any.
- **platforms** – Allowed OS names (`project.simos.name`, lowercased); None = any.
- **where** – Optional cross-capture predicate evaluated on the bindings after a structural match.
- **collapse_capture** – Optional capture name identifying the idiom that a match belongs to: within one function, matches binding it to the same virtual variable all describe *one* source-level idiom, and only the one with the largest `collapse_max_capture` constant is kept. The compiler folds CONTAINING_RECORD's subtraction into every field displacement, so one source idiom otherwise emits a call per field, each naming a different (and mostly wrong) record base.
- **collapse_max_capture** – Capture whose constant value orders the collapsing; required when `collapse_capture` is set.
- **binary_guard** – Optional predicate on the Project; the pattern is only used when it returns True (e.g. require C++ evidence for STL patterns to avoid false positives on plain C binaries).
- **default_enabled** – Whether the pattern is used when the caller does not explicitly select patterns. Generic patterns prone to false positives should set this False.

**name:** str

**display_name:** str

**call_name:** str

**pattern:** *PatternExpr* | *PatternStmt* | *PGraphPat*

**params:** tuple[*PatternParam*, ...]

**returnty:** str | *CppRef* | None = None

**returnty_factory:** Callable[[Arch, dict[str, int]], *SimType* | None] | None = None

**extra_args:** tuple[str, ...] = ()

**arches:** tuple[str, ...] | None = None

**platforms:** tuple[str, ...] | None = None

**where:** Callable[[dict[str, *Expression*]], bool] | None = None

**collapse_capture:** str | None = None

**collapse_max_capture:** str | None = None

**binary_guard:** `Callable[[Project], bool] | None = None`

**default_enabled:** `bool = True`

**suppressed_by:** `tuple[str, str, str] | None = None`

Drop a match of this pattern when a pattern whose name matches the regex matched on the same object: `(name regex, my capture, their capture)`. The two captures must bind `.likes()`-equal expressions. This is how a generic accessor yields to a specific one on the same base: a bare `Load(s + 8)` is std::string::length on a string and _M_finish on a vector, and the vector's other fields say which. Judged against every match of the suppressing family, selected or not, so it cannot interact with largest-match selection.

**claims_only:** `bool = False`

Match only to claim a base for `suppressed_by` purposes; never outlined, never rewritten, never reported. For shapes that identify an object beyond doubt but are not worth naming: a vector's capacity check says "this is a vector" and nothing a reader wants a call for.

**pure_calls:** `frozenset[str] = frozenset({})`

Callees this pattern reads through *PCallResult* whose only effect is their result: once the pattern has consumed that result and nothing else does, the call goes too. Naming a callee in a pattern is already a claim about what it computes; this is the claim that it computes nothing else. `__ctype_b_loc` returns a table pointer, and a bare `__ctype_b_loc();` left behind after `isspace(c)` is noise.

**applicable**(*arch_name*, *platform*)

**Return type**
bool

**Parameters**

- **arch_name** (*str*)
- **platform** (*str | None*)

**const_args_of_call**(*call*)

Recover the `extra_args` constant values from a synthesized call's trailing arguments. Returns None when the call does not carry them.

**Return type**
dict[str, int] | None

**Parameters**
**call** (Call)

**prototype**(*arch*, *const_args=None*)

Build the synthesized call's prototype. `const_args` (the values of the constant `extra_args` captures, see `const_args_of_call()`) enables call-site-specific return types via `returnty_factory`. Returns None if any declared type fails to resolve.

**Return type**
*SimTypeFunction* | None

**Parameters**

- **arch** (*Arch*)
- **const_args** (*dict[str, int] | None*)

**__init__**(*name*, *display_name*, *call_name*, *pattern*, *params*, *returnty=None*, *returnty_factory=None*, *extra_args=()*, *arches=None*, *platforms=None*, *where=None*, *collapse_capture=None*, *collapse_max_capture=None*, *binary_guard=None*, *default_enabled=True*, *suppressed_by=None*, *claims_only=False*, *pure_calls=frozenset({})*)

**Parameters**

- **name** (*str*)
- **display_name** (*str*)
- **call_name** (*str*)
- **pattern** (PatternExpr | PatternStmt | PGraphPat)
- **params** (*tuple[PatternParam, ...]*)
- **returnty** (*str | CppRef | None*)
- **returnty_factory** (*Callable[[Arch, dict[str, int]], SimType | None] | None*)
- **extra_args** (*tuple[str, ...]*)
- **arches** (*tuple[str, ...] | None*)
- **platforms** (*tuple[str, ...] | None*)
- **where** (*Callable[[dict[str, Expression]], bool] | None*)
- **collapse_capture** (*str | None*)
- **collapse_max_capture** (*str | None*)
- **binary_guard** (*Callable[[Project], bool] | None*)
- **default_enabled** (*bool*)
- **suppressed_by** (*tuple[str, str, str] | None*)
- **claims_only** (*bool*)
- **pure_calls** (*frozenset[str]*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**KnownPatternFinder**

Bases: *Analysis*

Finds occurrences of KnownPatterns in a Clinic AIL graph.

The graph to match on is the final (SSA-form) Clinic graph, i.e. `decompiler.ail_graph` / `clinic.cc_graph`. Matches are reported in `self.matches`; `outline()` turns a match into a synthesized call by invoking the Outliner analysis on a copy of the graph.

**MAX_REMOTE_CHASE_DEPTH = 6**

**__init__**(*func*, *ail_graph*, *patterns=None*, *chase_defs=True*, *chase_remote_defs=True*, *skip_conversions=True*, *vvar_id_start=48879*, *block_addr_start=2864381952*, *ail_manager=None*, *force_patterns=None*)

`patterns` may be pattern templates (KnownPatternTemplate) and/or already-concrete KnownPatterns; templates are instantiated for this binary's PatternContext. When None, the registry is used, filtered down to the templates that are enabled for this target: default-on ones, ones named by `force_patterns`, and ones whose gate opens (see *gating*).

`force_patterns` is the user's force-enable selection, `"all"` or an iterable of template names / call names, and is only consulted when `patterns` is None (an explicit pattern list is already a selection).

**Parameters**

- **func** (Function)
- **ail_graph** (*DiGraph*)
- **patterns** (*Iterable[object] | None*)
- **chase_defs** (*bool*)
- **chase_remote_defs** (*bool*)
- **skip_conversions** (*bool*)
- **vvar_id_start** (*int*)
- **block_addr_start** (*int*)
- **force_patterns** (*str | Iterable[str] | None*)

**matches:** list[*KnownPatternMatch*]

**call_target_names**(*call*)

Public face of `_resolve_call_target()`, for the outliner's clean-up of calls a pattern declared pure.

**Return type**
frozenset[str]

**Parameters**
**call** (Call)

**apply_matches**(*matches*, *ail_graph*)

Apply as many of `matches` as possible, batching where it matters.

Applying one match at a time is fine until several of them interleave in one block, which is exactly what consecutive field swaps do:

**t1 = x->b; t2 = y->b; y->b = t1; x->b = t2;**
^ the next field's load sits in this one's gap

Rewrite the first and the second's statements have moved; rewrite it anyway and the third now has an opaque call sitting in its gap, which no disjointness test will let it move across. So the void statement-span matches of a block, which the finder has already checked do not conflict, are spliced in together, against the statement indices they were found at. Everything else keeps the one-at-a-time path.

Returns the new graph and the ids of the matches that were applied.

**Return type**
tuple[DiGraph, set[int]]

**Parameters**

- **matches** (*list[KnownPatternMatch]*)
- **ail_graph** (*DiGraph*)

**outline**(*match*, *ail_graph=None*)

Outline one match: split the anchor block so the matched span forms its own block, invoke the Outliner analysis on it, and rewrite the synthesized callsite into the pattern's call. Operates on (and returns) a copy; neither `self._graph` nor `ail_graph` is mutated.

**Return type**
*OutlineResult*

**Parameters**

- **match** (KnownPatternMatch)

- **ail_graph** (*DiGraph | None*)

**MAX_LIFTED_EXPR_NODES = 8**

Node budget for a lifted parameter expression; see _is_liftable.

**outline_at**(*location*, *pattern*, *ail_graph*)

Find `pattern` at `location` (`(block_addr, block_idx)` or `(block_addr, block_idx, stmt_idx)`) in `ail_graph` and outline it.

**Return type**
*OutlineResult*

**Parameters**

- **location** (*tuple[int, int | None] | tuple[int, int | None, int]*)
- **pattern** (KnownPattern)
- **ail_graph** (*DiGraph*)

**class** angr.analyses.decompiler.known_patterns.**KnownPatternMatch**

Bases: object

One occurrence of a KnownPattern in an AIL graph.

For expression-level matches, the boundary is `consumed_stmt_idxs` (statements wholly owned by the match, e.g. chased definitions) plus `expr_path` (the location of the matched sub-expression inside the anchor statement). For statement-sequence matches, `stmt_span` holds the ordered matched statement indices (the anchor is the first) and `matched_expr` is None.

`duplicated_stmt_idxs` holds chased definitions that are copied into the outlined region instead of moved: the originals stay where they are (other code still uses them) and the copies make the pattern's captured values the region's live-ins. It is always disjoint from `consumed_stmt_idxs`.

`recomputed_defs` does the same for definitions that live in *another* block (a CSE'd value, possibly reaching the anchor through a phi): the expression is pure, so the region recomputes it from the captured operands.

**pattern:** *KnownPattern*

**block_loc:** tuple[int, int | None]

**anchor_stmt_idx:** int

**expr_path:** tuple[tuple[str, int | None], ...]

**consumed_stmt_idxs:** frozenset[int]

**captures:** dict[str, *Expression*]

**matched_expr:** *Expression* | None

**duplicated_stmt_idxs:** frozenset[int] = frozenset({})

**recomputed_defs:** tuple[tuple[int, *Expression*], ...] = ()

**base_aliases:** tuple[tuple[int, int, int], ...] = ()

**stmt_span:** tuple[int, ...] | None = None

**block_map:** dict[str, tuple[int, int | None]] | None = None

**consumed_by_block:** dict[tuple[int, int | None], frozenset[int]] | None = None

**frontier_locs**: frozenset[tuple[int, int | None]] | None = None

**__init__**(*pattern*, *block_loc*, *anchor_stmt_idx*, *expr_path*, *consumed_stmt_idxs*, *captures*, *matched_expr*, *duplicated_stmt_idxs=frozenset({})*, *recomputed_defs=()*, *base_aliases=()*, *stmt_span=None*, *block_map=None*, *consumed_by_block=None*, *frontier_locs=None*)

**Parameters**

- **pattern** (KnownPattern)
- **block_loc** (*tuple[int, int | None]*)
- **anchor_stmt_idx** (*int*)
- **expr_path** (*tuple[tuple[str, int | None], ...]*)
- **consumed_stmt_idxs** (*frozenset[int]*)
- **captures** (*dict[str, Expression]*)
- **matched_expr** (*Expression | None*)
- **duplicated_stmt_idxs** (*frozenset[int]*)
- **recomputed_defs** (*tuple[tuple[int, Expression], ...]*)
- **base_aliases** (*tuple[tuple[int, int, int], ...]*)
- **stmt_span** (*tuple[int, ...] | None*)
- **block_map** (*dict[str, tuple[int, int | None]] | None*)
- **consumed_by_block** (*dict[tuple[int, int | None], frozenset[int]] | None*)
- **frontier_locs** (*frozenset[tuple[int, int | None]] | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**KnownPatternTemplate**

Bases: object

A context-aware pattern definition.

**Variables**

- **call_name** – The synthesized call name (unique across templates).
- **build** – `build(ctx) -> KnownPattern | None`; emits the concrete pattern for `ctx` (None if it cannot).
- **arches** – Allowed `Arch.name` values; None = any.
- **languages** – Allowed languages ({"c","cpp"}); None = any.
- **runtimes** – Allowed C++ runtimes ({"libstdcxx","msvc"}); None = any.
- **platforms** – Allowed platforms ({"linux","windows"}); None = any.
- **default_enabled** – Whether the template is used when the caller does not explicitly select patterns.
- **gate** – Optional *PatternGate* that turns an opt-in template on for targets (or functions) where the evidence says the idiom is real; see *gating*. A gate can only open a template, never close a default-on one.

**call_name:** *str*

**build:** *Callable[[PatternContext], KnownPattern | None]*

**arches:** *frozenset[str] | None* = None

**languages:** *frozenset[str] | None* = None

**runtimes:** *frozenset[str] | None* = None

**platforms:** *frozenset[str] | None* = None

**default_enabled:** *bool* = False

**name:** *str* = ''

**gate:** *PatternGate | None* = None

**enabled_for**(*gate_ctx*)

**Return type**
bool

**Parameters**
**gate_ctx** (GateContext)

**applicable**(*ctx*)

**Return type**
bool

**Parameters**
**ctx** (PatternContext)

**instantiate**(*ctx*)

The concrete pattern for `ctx`, or None if this template does not apply there. Memoized: a KnownPattern is a frozen tree that depends on nothing but the context, and callers treat it read-only.

**Return type**
*KnownPattern* | None

**Parameters**
**ctx** (PatternContext)

**__init__**(*call_name*, *build*, *arches=None*, *languages=None*, *runtimes=None*, *platforms=None*, *default_enabled=False*, *name=''*, *gate=None*, *_cache=<factory>*)

**Parameters**

- **call_name** (*str*)
- **build** (*Callable[[PatternContext], KnownPattern | None]*)
- **arches** (*frozenset[str] | None*)
- **languages** (*frozenset[str] | None*)
- **runtimes** (*frozenset[str] | None*)
- **platforms** (*frozenset[str] | None*)
- **default_enabled** (*bool*)
- **name** (*str*)

- **gate** (PatternGate | *None*)
- **_cache** (*dict*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**OutlineResult**

Bases: object

The outcome of outlining one KnownPatternMatch.

**graph:** DiGraph

**match:** *KnownPatternMatch*

**call_stmt:** *Statement*

**child_func:** *Function* | None

**child_graph:** DiGraph | None

**child_funcargs:** list[*VirtualVariable*]

**__init__**(*graph*, *match*, *call_stmt*, *child_func*, *child_graph*, *child_funcargs*)

**Parameters**

- **graph** (*DiGraph*)
- **match** (KnownPatternMatch)
- **call_stmt** (Statement)
- **child_func** (Function | *None*)
- **child_graph** (*DiGraph* | *None*)
- **child_funcargs** (*list*[VirtualVariable])

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PAny**

Bases: *PatternExpr*

Matches any expression.

**name:** str | None = None

**bits:** int | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

__init__(*name=None*, *bits=None*)

**Parameters**

- **name** (*str | None*)
- **bits** (*int | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PAssign**

Bases: *PatternStmt*

Matches an Assignment statement.

**dst:** *PatternExpr*

**src:** *PatternExpr*

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

__init__(*dst*, *src*)

**Parameters**

- **dst** (PatternExpr)
- **src** (PatternExpr)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PBinOp**

Bases: *PatternExpr*

Matches a BinaryOp. op may be a single op string or a set of acceptable ops. Commutative ops try both operand orders unless commutative is explicitly False.

**op:** `str | frozenset[str]`

**operands:** `tuple[PatternExpr, PatternExpr]`

**commutative:** `bool | None = None`

**name:** `str | None = None`

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)

- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*op*, *operands*, *commutative=None*, *name=None*)

**Parameters**

- **op** (*str | frozenset[str]*)
- **operands** (*tuple[PatternExpr, PatternExpr]*)
- **commutative** (*bool | None*)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**PBlockPat**

Bases: *PatternNode*

A labeled block of statements in a multi-block pattern graph. An empty `stmts` sequence matches any block (structure alone identifies it).

**label:** `str`

**stmts:** *PStmtSeq*

**__init__**(*label*, *stmts*)

**Parameters**

- **label** (*str*)
- **stmts** (PStmtSeq)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**PCall**

Bases: *PatternExpr*

Matches a `Call` whose callee is one of `names`.

The callee is what makes a pattern containing this node self-guarding: an idiom spelled around a named runtime function (`operator delete` in the std::string destructor, `__ctype_b_loc` in `isspace`) cannot be confused with arithmetic that happens to look alike, because no other code calls that function to do something else.

`names` is matched against every spelling the callee is known by (the raw symbol and its demangled form), so a pattern may name either. Prefer the mangled spelling: it is exact, while a demangled one carries the argument list and varies with the demangler.

`args`=`None` leaves the arguments unconstrained; a tuple constrains them positionally, with `None` in a slot meaning "any expression".

**names:** `frozenset[str]`

**args:** `tuple[PatternExpr | None, ...] | None = None`

**name:** `str | None = None`

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*names*, *args=None*, *name=None*)

**Parameters**

- **names** (*frozenset[str]*)
- **args** (*tuple[PatternExpr | None, ...] | None*)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**PCallResult**

Bases: *PatternExpr*

The value a call to one of `names` returned.

A call's result is rarely used where it is produced: the compiler assigns it to a register and reads that register, often in another block. `__ctype_b_loc()` is called once per function and its table indexed at every predicate. So this node matches the virtual variable and looks up its definition read-only. Nothing is consumed, moved or recomputed; the call stays where the compiler put it.

An inline `Call` expression is matched too, for the case where the value is used at its definition.

**names:** `frozenset[str]`

**name:** `str | None = None`

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*names*, *name=None*)

**Parameters**

- **names** (*frozenset[str]*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PCallStmt**

Bases: *PatternStmt*

Matches a statement whose effect is a call.

Ailment has no Call statement: a call whose result is discarded is a `SideEffectStatement` wrapping the Call expression, and one whose result is kept is an ordinary `Assignment`. Both spellings are the same idiom, so both match; `dst`, when given, also constrains the assigned destination, which requires the assignment form.

**call:** *PCall*

**dst:** *PatternExpr* | None = None

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*call*, *dst=None*)

**Parameters**

- **call** (PCall)
- **dst** (PatternExpr | *None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PChoice**

Bases: *PatternExpr*

Ordered alternation: matches the first alternative that succeeds.

**alternatives:** tuple[*PatternExpr*, ...]

**__init__**(**alternatives*)

**Parameters**
**alternatives** (PatternExpr)

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

class angr.analyses.decompiler.known_patterns.**PCondJump**

Bases: *PatternStmt*

Matches a ConditionalJump, matching its condition expression.

**condition:** *PatternExpr*

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*condition*)

**Parameters**
**condition** (PatternExpr)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PConst**

Bases: *PatternExpr*

Matches a Const, by exact value or by predicate. Width is ignored unless `bits` is given (e.g. shift amounts are often 8-bit).

**value:** `int | None = None`

**pred:** `Callable[[int], bool] | None = None`

**bits:** `int | None = None`

**name:** `str | None = None`

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*value=None*, *pred=None*, *bits=None*, *name=None*)

**Parameters**

- **value** (*int | None*)
- **pred** (*Callable[[int], bool] | None*)
- **bits** (*int | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PConv**

Bases: *PatternExpr*

Matches a Convert explicitly (never skipped).

**operand:** *PatternExpr*

**from_bits:** int | None = None

**to_bits:** int | None = None

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*operand*, *from_bits=None*, *to_bits=None*, *name=None*)

**Parameters**

- **operand** (PatternExpr)
- **from_bits** (*int | None*)
- **to_bits** (*int | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PDefOf**

Bases: *PatternExpr*

`inner`, or a virtual variable whose reaching definition is `inner`.

The compiler is free to compute a value once and use it twice, and it does so exactly where an idiom uses one value twice. `if (_M_p != &_M_local_buf) operator delete(_M_p, ...)` reads _M_p in the compare and in the call, so gcc loads it into a register and the pattern sees `Load(s)` in neither place; it sees the same virtual variable in both.

Reading that variable's definition is read-only. Nothing is consumed, moved or recomputed, and the identification stays accurate even if memory changes afterwards: the variable holds what the load produced at the definition, which is what the idiom meant. That is why this is not `peek_fn`, whose contract ("can be recomputed at the use") a load cannot satisfy.

The unconsumed definition is left where it was, so a region that matches through this node keeps it as residue. That is harmless, and dead-code elimination removes it once the idiom's other uses are gone.

**inner:** *PatternExpr*

**name:** *str | None = None*

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*inner*, *name=None*)

**Parameters**

- **inner** (PatternExpr)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**PExtract**

Bases: *PatternExpr*

Matches an `Extract`, a bit-slice of a wider value.

This is how a scalar reaches integer code out of a vector register when the 128-bit op did not get narrowed away: `movq %xmm2,%rax` on a value that came from `maxsd`/`mulsd` lifts to `Extract(vvar_128, 64bits@0)` rather than to a Convert. Without a node for it, every libm bit-twiddle whose operand is a computed double rather than an incoming argument is unmatchable.

`bits`/`offset` default to unconstrained; `offset` is in bits.

**operand:** *PatternExpr*

**bits:** *int | None = None*

**offset:** *int | None = None*

**name:** *str | None = None*

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*operand*, *bits=None*, *offset=None*, *name=None*)

**Parameters**

- **operand** (PatternExpr)
- **bits** (*int | None*)

- **offset** (*int | None*)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**PField**

Bases: *PatternExpr*

The *address* of the field at `offset` inside the object bound to `base`.

Spelling a container's fields as `PLoad(PVVar("v"))` / `PLoad(PVVar("v") + 8)` requires the object to sit at the very address held in a virtual variable. That is the minority case: measured over the benchmark corpus, only 10-25% of the DWARF sites for a `std::vector` accessor read the container through a bare pointer; the rest reach it as a *field of something else* (`this->tokens.size()`), which lifts to `Load(this + 56)` / `Load(this + 64)` and cannot bind `v` at all.

`PField` matches such an address and binds `base` to the object's own address: `root` when the object is at offset 0, else `root + K`. Two fields of one object therefore unify on `K`, which is what keeps the node honest: it replaces the constraint "this displacement is exactly 8" with "these two displacements differ by exactly 8".

Only use it in patterns that reference at least two fields. With a single field there is no second displacement to constrain, so `PLoad(PField("s", 8))` degenerates into "any load through a pointer plus a non-negative constant" and matches essentially everything.

`root` must be a virtual variable: the outliner materializes `tmp = root + K` ahead of the region so the synthesized call still takes a variable.

**base:** *str*

**offset:** *int*

**name:** *str | None* = **None**

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (*Expression*)
- **state** (*MatchState*)
- **ctx** (*MatchCtx*)

**__init__**(*base*, *offset*, *name=None*)

**Parameters**

- **base** (*str*)
- **offset** (*int*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PGraphPat**

Bases: *PatternNode*

A multi-block pattern: labeled blocks connected by edges.

`blocks` maps a label to its `PBlockPat`. `edges` are `(src_label, dst_label)` pairs; a destination label that does not appear in `blocks` denotes an external successor, a region exit that becomes the Outliner frontier. `entry` is the label of the single entry block. Captures unify across all blocks (one bindings environment).

**blocks:** `dict[str, PBlockPat]`

**edges:** `Sequence[tuple[str, str]]`

**entry:** `str`

**property external_labels:** `set[str]`

**__init__**(*blocks*, *edges*, *entry*)

**Parameters**

- **blocks** (*dict[str, PBlockPat]*)
- **edges** (*Sequence[tuple[str, str]]*)
- **entry** (*str*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PLoad**

Bases: *PatternExpr*

Matches a memory Load. `size` is in bytes.

**addr:** *PatternExpr*

**size:** `int | None = None`

**name:** `str | None = None`

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*addr*, *size=None*, *name=None*)

**Parameters**

- **addr** (PatternExpr)
- **size** (*int | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PPhi**

Bases: *PatternExpr*

Matches a Phi expression, capturing it whole (its individual sources are not matched). Useful for loop-header blocks, whose statements are phi assignments.

**name:** str | None = None

**bits:** int | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*name=None*, *bits=None*)

**Parameters**

- **name** (*str | None*)
- **bits** (*int | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PStackField**

Bases: *PatternExpr*

A field of a container that lives on the stack.

A stack object never reaches the matcher as an object at all: variable recovery has already split it into one virtual variable per slot, so `std::string s;` is a handful of independent `vvar{s-112}`, `vvar{s-96}` and there is no `Load(s)` to match and no `s + 16` to compare against. Measured over the benchmark corpus (*kp-eval/stack_share.py*), stack objects are 40-54% of the DWARF sites for `std::string::length` and 15-35% for `std::vector<T>::size`.

This node matches the slot of the field at `offset` and binds `base` to the object's own stack offset, so two fields of one object unify on it exactly as `PField`'s two displacements do. `as_address` matches the address form (`&slot`, which lifts to `UnaryOp("Reference", vvar)`) rather than the value form; `std::string::capacity` needs both, since it compares the data pointer against the address of the local buffer.

The binding is a `StackBaseOffset`, the object's own address. That is both the thing two fields must agree on and, directly, the argument the synthesized call takes, so nothing has to be materialized for it.

Matching is only half the problem, and the other half is why a pattern using this node is never outlined: the region for a stack container reads N independent slots, so its live-ins are N values rather than one pointer and there is nothing for a synthesized callee to take. Such a match is instead rewritten in place: the matched expression, or the whole region, is replaced by `call(&s)`, whose argument the binding already is. See `KnownPatternFinder._rewrite_in_place` and `_rewrite_graph_in_place`.

**base:** str

**offset:** int

**as_address:** bool = False

**size:** int | None = None

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*base*, *offset*, *as_address=False*, *size=None*, *name=None*)

**Parameters**

- **base** (*str*)
- **offset** (*int*)
- **as_address** (*bool*)
- **size** (*int | None*)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**PStmtSeq**

Bases: *PatternStmt*

Matches a group of statements within one block. When `ordered` (the default), the statement patterns must match in order; with `allow_gaps`, unrelated statements may sit between the matched ones. When `ordered` is False, the statement patterns match statements of the block in any order (each pattern to a distinct statement), which suits idioms whose statement order the compiler chooses freely (e.g. list-link stores).

**stmts:** tuple[*PatternStmt*, ...]

**allow_gaps:** bool = True

**ordered:** bool = True

**max_gap:** int = 8

How many unmatched statements may sit between two matched ones. Without a bound the scan runs to the end of the block from every statement it could start at, quadratic per block per template, and finds nothing: an idiom's statements are adjacent apart from what the scheduler interleaved, which is a handful of instructions rather than a basic block.

**__init__**(*stmts*, *allow_gaps=True*, *ordered=True*, *max_gap=8*)

**Parameters**

- **stmts** (*tuple[PatternStmt, ...]*)
- **allow_gaps** (*bool*)
- **ordered** (*bool*)

- **max_gap** (*int*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PStore**

Bases: *PatternStmt*

Matches a Store statement. `size` is in bytes.

**addr:** *PatternExpr*

**value:** *PatternExpr*

**size:** `int | None = None`

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*addr*, *value*, *size=None*)

**Parameters**

- **addr** (PatternExpr)
- **value** (PatternExpr)
- **size** (*int | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PUnaryOp**

Bases: *PatternExpr*

Matches a UnaryOp.

**op:** `str | frozenset[str]`

**operand:** *PatternExpr*

**name:** `str | None = None`

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*op*, *operand*, *name=None*)

**Parameters**

- **op** (*str | frozenset[str]*)
- **operand** (PatternExpr)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PVVar**

Bases: *PatternExpr*

Matches a VirtualVariable.

**name:** str | None = None

**bits:** int | None = None

**categories:** frozenset[VirtualVariableCategory] | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*name=None*, *bits=None*, *categories=None*)

**Parameters**

- **name** (*str | None*)
- **bits** (*int | None*)
- **categories** (*frozenset[VirtualVariableCategory] | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.**PatternContext**

Bases: object

Target facts used to instantiate pattern templates.

The last two fields are binary evidence: facts about what kind of program this is, used by *gating* to switch whole pattern families on for targets where the idioms are certain to appear and off everywhere else.

**arch_name:** str

**bits:** int

**ptr_size:** int

**platform:** str | None

**cxx_runtime:** *str | None*

**is_cpp:** *bool*

**is_linux_kernel_object:** *bool = False*

**is_windows_kernel_driver:** *bool = False*

**classmethod from_project**(*project*)

**Return type**
*PatternContext*

**Parameters**
**project** (Project)

**word**(*n*)

Byte offset of the n-th pointer-word field.

**Return type**
int

**Parameters**
**n** (*int*)

**property word_size:** *int*

Load size (bytes) of a pointer / size_t field.

**property language:** *str*

**__init__**(*arch_name*, *bits*, *ptr_size*, *platform*, *cxx_runtime*, *is_cpp*, *is_linux_kernel_object=False*, *is_windows_kernel_driver=False*)

**Parameters**

- **arch_name** (*str*)
- **bits** (*int*)
- **ptr_size** (*int*)
- **platform** (*str | None*)
- **cxx_runtime** (*str | None*)
- **is_cpp** (*bool*)
- **is_linux_kernel_object** (*bool*)
- **is_windows_kernel_driver** (*bool*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**PatternGate**

Bases: object

Base class for template gates. Instances are callables so that a gate can be used anywhere a `Callable[[GateContext], bool]` is expected.

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence: bool**

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label: str**

**__init__()**

**Return type**
None

**exception** angr.analyses.decompiler.known_patterns.**PatternGenerationError**

Bases: Exception

Raised when a selection cannot be turned into a KnownPattern (ambiguous, unsupported, or inconsistent with the given arguments).

**class** angr.analyses.decompiler.known_patterns.**PatternGenerator**

Bases: object

Generate a KnownPattern from a text selection in codegen.text.

**Parameters**

- **codegen** – the Decompiler.codegen (a CStructuredCodeGenerator).
- **ail_graph** (DiGraph | None) – the graph the generated pattern will be matched against (Decompiler.ail_graph). Required for multi-block (control-flow-spanning) selections; optional otherwise.

**__init__**(*codegen*, *ail_graph=None*)

**Parameters**
**ail_graph** (*DiGraph | None*)

**vvar_id_at**(*offset*)

The AIL vvar id of the variable rendered at offset, or None.

**Return type**
int | None

**Parameters**
**offset** (*int*)

**offset_of**(*needle*, *start=0*)

Convenience: the offset of needle in the rendered text.

**Return type**
int

**Parameters**

- **needle** (*str*)
- **start** (*int*)

**generate**(*start_offset*, *end_offset*, *call_name*, *arg_offsets*, ***, *name=None*, *display_name=None*, *returnty=None*, *param_types=None*, *const_arg_offsets=()*, *arches=None*, *platforms=None*, *binary_guard=None*, *default_enabled=False*)

**Return type**
*KnownPattern*

**Parameters**

- **start_offset** (*int*)
- **end_offset** (*int*)
- **call_name** (*str*)
- **arg_offsets** (*Sequence[int]*)
- **name** (*str | None*)
- **display_name** (*str | None*)
- **returnty** (*str | CppRef | None*)
- **param_types** (*Sequence[str | CppRef | None] | None*)
- **const_arg_offsets** (*Sequence[int]*)
- **arches** (*tuple[str, ...] | None*)
- **platforms** (*tuple[str, ...] | None*)
- **default_enabled** (*bool*)

class angr.analyses.decompiler.known_patterns.**PatternParam**

Bases: object

One parameter of the synthesized call.

**Variables**

- **capture** – Name of the pattern capture bound to this parameter.
- **type** – Argument type for the synthesized call's prototype; flows into Typehoon as a per-argument subtype constraint.
- **type_hint** – Optional cpp::std unique name emitted as a direct vvar type hint (for by-value captures where the vvar *is* the object rather than a pointer to it).

**capture:** str

**type:** str | *CppRef* | None = None

**type_hint:** str | None = None

**__init__**(*capture*, *type=None*, *type_hint=None*)

**Parameters**

- **capture** (*str*)
- **type** (*str | CppRef | None*)
- **type_hint** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.**TargetGate**

Bases: *PatternGate*

Opens when a predicate over the target's PatternContext holds.

The predicate reads facts that were derived once from the loaded binary, so evaluating it is free and its result is the same for every function.

**name:** *str*

**predicate:** *Callable[[PatternContext], bool]*

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property label:** *str*

**__init__**(*name*, *predicate*)

**Parameters**

- **name** (*str*)
- **predicate** (*Callable[[PatternContext], bool]*)

**Return type**
None

**exception** angr.analyses.decompiler.known_patterns.**UnknownPatternError**

Bases: ValueError

Raised when a pattern selection names a template that is not registered.

**exception** angr.analyses.decompiler.known_patterns.**UnsupportedOutlineError**

Bases: Exception

Raised when a match cannot be safely outlined (e.g. side-effecting statements interleave with the matched span, or the recovered callee interface does not agree with the pattern's declared parameters).

angr.analyses.decompiler.known_patterns.**all_of**(**gates*)

**Return type**
*AllOf*

**Parameters**
**gates** (PatternGate)

angr.analyses.decompiler.known_patterns.**any_of**(**gates*)

**Return type**
*AnyOf*

**Parameters**
**gates** (PatternGate)

angr.analyses.decompiler.known_patterns.**corroborated_by**(**witnesses*)

A gate that opens in a function where one of `witnesses` matched.

**Return type**
*CorroboratedBy*

**Parameters**
**witnesses** (*str*)

angr.analyses.decompiler.known_patterns.**exact_div_magic**(*elt_size*, *bits*)

`(shift, magic)` of the exact-division-by-`elt_size` idiom.

**Return type**
tuple[int, int]

**Parameters**

- **elt_size** (*int*)
- **bits** (*int*)

angr.analyses.decompiler.known_patterns.**make_std_vector_size_exactdiv_template**(*elt_name*, *elt_size*)

A non-power-of-two `std::vector<elt_name>::size` template.

**Parameters**

- **elt_name** (*str*)
- **elt_size** (*int*)

angr.analyses.decompiler.known_patterns.**make_std_vector_size_template**(*elt_name*, *log2_elt_size*)

A power-of-two `std::vector<elt_name>::size` template.

**Parameters**

- **elt_name** (*str*)
- **log2_elt_size** (*int*)

angr.analyses.decompiler.known_patterns.**make_template**(*call_name*, *build*, ***, *arches=None*, *languages=None*, *runtimes=None*, *platforms=None*, *default_enabled=False*, *name=''*, *gate=None*)

**Return type**
*KnownPatternTemplate*

**Parameters**

- **call_name** (*str*)
- **build** (*Callable[[PatternContext], KnownPattern | None]*)
- **arches** (*frozenset[str] | tuple[str, ...] | None*)
- **languages** (*frozenset[str] | tuple[str, ...] | None*)
- **runtimes** (*frozenset[str] | tuple[str, ...] | None*)
- **platforms** (*frozenset[str] | tuple[str, ...] | None*)
- **default_enabled** (*bool*)
- **name** (*str*)

- **gate** (PatternGate | *None*)

angr.analyses.decompiler.known_patterns.**partition_templates**(*gate_ctx*, *forced=()*, *templates=None*)

Split templates into `(enabled, deferred)` for one target.

`enabled` are the templates to match with right away: default-on ones, ones the caller force-enabled, and ones whose gate already opens on target evidence alone. `deferred` are the ones whose gate needs per-function evidence (see *CorroboratedBy*) and must be re-evaluated by the finder once the first matching stage has run.

**Return type**
tuple[list[*KnownPatternTemplate*], list[*KnownPatternTemplate*]]

**Parameters**

- **gate_ctx** (GateContext)
- **forced** (*Iterable*[KnownPatternTemplate])
- **templates** (*Iterable*[KnownPatternTemplate] | *None*)

angr.analyses.decompiler.known_patterns.**patterns_for**(*ctx*, *templates=None*, ***, *enabled_only=False*)

Instantiate the applicable templates for `ctx` into concrete KnownPatterns. `templates` defaults to the whole registry; with `enabled_only` only default-enabled templates are used.

**Return type**
list[*KnownPattern*]

**Parameters**

- **ctx** (PatternContext)
- **templates** (*Iterable*[KnownPatternTemplate] | *None*)
- **enabled_only** (*bool*)

angr.analyses.decompiler.known_patterns.**register_pattern_template**(*template*)

Register a KnownPatternTemplate. Call names are unique across templates (the architecture / runtime conditionals live inside each template's `build`).

**Return type**
None

**Parameters**
**template** (KnownPatternTemplate)

angr.analyses.decompiler.known_patterns.**resolve_pattern_selection**(*selection*)

Resolve a user's force-enable selection into templates.

`selection` is one of:

- `None` / empty: nothing is force-enabled (returns an empty list);
- `"all"`: every registered template, opt-in ones included;
- an iterable of template ``name``s and/or ``call_name``s (a comma-separated string is accepted too, for the string-valued decompilation option).

An unrecognized name raises `UnknownPatternError` rather than silently selecting nothing.

**Return type**
list[*KnownPatternTemplate*]

**Parameters**
**selection** (*str* | *Iterable*[str] | *None*)

### Submodules

| | |
|---|---|
| *apply* | Apply call-site information (prototypes) for known-pattern calls. |
| *block_split* | Utilities for splitting an AIL block into consecutive blocks within a graph. |
| *containing_record* | The CONTAINING_RECORD macro (Windows drivers) / container_of (Linux kernel): |
| *context* | PatternContext: the architecture / platform / language facts a KnownPattern template needs to instantiate itself for a given binary. |
| *ctype_tables* | The glibc ctype macros: isspace, isdigit, . |
| *dsl* | Declarative pattern-AST for describing known AIL code idioms (KnownPatterns). |
| *finder* | KnownPatternFinder: find occurrences of KnownPatterns in a Clinic AIL graph and outline them into calls via the Outliner analysis. |
| *gating* | Criteria gates: per-target (and per-function) enablement for pattern templates. |
| *generator* | PatternGenerator: build a KnownPattern from a decompilation text selection. |
| *kernel_err* | Linux kernel <`linux/err.h`> error-pointer idioms. |
| *layouts* | Runtime-specific struct layout offsets for the STL container patterns. |
| *libm_bits* | libm floating-point bit-twiddling idioms (fabs / -x / copysign / isnan / isinf). |
| *linked_list* | Doubly-linked-list idioms shared by Windows `LIST_ENTRY` (Flink/Blink) and Linux kernel `list_head` (next/prev): the same two-pointer layout, so one set of templates matches both WDK drivers and kernel/Boost-intrusive code. |
| *pattern* | KnownPattern: declarative metadata around a pattern-AST. |
| *posix_macros* | Single-expression glibc/POSIX macros that the preprocessor inlines into every caller, so they survive into decompiled output as bare bit arithmetic. |
| *protobuf_hasbits* | protobuf has-bits accessors (protoc output for optional fields): |
| *registry* | The template registry: every `KnownPatternTemplate` the package defines, keyed by call name and by human name, plus the helpers that turn a user's selection or a target's gates into the list of templates to match with. |
| *std_string_cstr* | Inlined MSVC `std::string::c_str()` / `data()`: the small-string optimization (SSO) select. |
| *std_string_dtor* | The inlined `std::string` destructor. |
| *std_string_internals* | libstdc++ std::string internals that are not accessors: the SSO test and the two length-setting idioms. |
| *std_string_length* | Inlined std::string::length() / size() and empty(). |
| *std_swap* | Inlined `std::swap`, the three-move exchange of two memory locations. |
| *std_vector_size* | Inlined std::vector<T>::size() and capacity(). |
| *stl_accessors2* | C++ STL accessors, round two: std::string capacity/back/front and std::vector<T>::back(). |

continues on next page

Table 15 – continued from previous page

| | |
|---|---|
| *stl_containers* | C++ STL std::vector accessors beyond size()/capacity(): empty / operator[]. |
| *templates* | KnownPatternTemplate: a single, context-parameterized definition of an idiom that instantiates itself into a concrete `KnownPattern` for a target's `PatternContext`. |
| *vector_claims* | Claims-only std::vector fingerprints: shapes that identify a vector beyond doubt but are not worth a call of their own. |
| *vector_math* | 3-component float vector math (glm / DirectXMath / game-graphics code). |
| *wdk_shared_data* | Reads of `KUSER_SHARED_DATA`, the page Windows maps read-only at the fixed virtual address `0x7FFE0000` in every user-mode process (x86, x64 and ARM64 alike). ntdll, kernel32 and countless drivers read its fields through the `SharedUserData` macro of nt-ddk.h/wdm.h, which compiles to a bare load from an absolute constant address::. |

### angr.analyses.decompiler.known_patterns.apply

Apply call-site information (prototypes) for known-pattern calls.

Known-pattern calls are identified by their string target, which is stable across Clinic runs; the VariableMap holding prototypes is scoped to a single Clinic run. This helper (re)applies the prototypes onto a graph's calls so that variable recovery emits per-argument type constraints for Typehoon. It is used by the KnownPatternOutliner optimization pass (which runs after the peephole stages) and mirrors what the KnownPatternCallInfo peephole does for graphs that already contain known-pattern calls at the start of a run.

`angr.analyses.decompiler.known_patterns.apply.`**`prototype_for_call`**(*project*, *call*)

The ctx-instantiated prototype for a known-pattern call, or None. The return-type width and argument pointer sizes follow the target's PatternContext.

**Parameters**

- **project** (Project)
- **call** (Call)

`angr.analyses.decompiler.known_patterns.apply.`**`apply_call_info_to_graph`**(*graph*, *manager*, *project*)

Set the call-site prototype of every known-pattern call in `graph` in the VariableMap attached to `manager`. Returns the number of calls that received a prototype.

**Return type**
int

**Parameters**

- **graph** (*networkx.DiGraph*)
- **manager** (*Manager*)
- **project** (Project)

### angr.analyses.decompiler.known_patterns.block_split

Utilities for splitting an AIL block into consecutive blocks within a graph.

The Outliner analysis operates at whole-block granularity, while KnownPattern matches are usually (sub-)statement-level; splitting the anchor block so the matched span becomes its own block bridges the two.

angr.analyses.decompiler.known_patterns.block_split.**rewrite_phi_sources**(*block*, *old_src*, *new_src*, *idx_alloc*)

Rewrite phi statements at the top of `block` so that entries sourced from `old_src` are sourced from `new_src` instead. Mutates the block's statement list in place.

`idx_alloc` supplies AIL indices for the rebuilt statements: an object built with `idx=None` lands at index 0, and the decompiler's VariableMap is keyed by index, so all of them would share one entry.

**Return type**
None

**Parameters**

- **block** (Block)
- **old_src** (*tuple[int, int | None]*)
- **new_src** (*tuple[int, int | None]*)
- **idx_alloc** (*Callable[[], int]*)

angr.analyses.decompiler.known_patterns.block_split.**split_ail_block**(*graph*, *block*, *pre_stmts*, *mid_stmts*, *post_stmts*, *block_addr_alloc*, *idx_alloc*)

Replace `block` in `graph` with up to three consecutive blocks holding `pre_stmts`, `mid_stmts`, and `post_stmts`.

The first non-empty part keeps the original block identity `(addr, idx)` so that predecessor jump targets (and the at-most-one-block-per-address assumption at the function entry) remain valid; each following part is placed at a fresh synthetic address obtained from `block_addr_alloc`. When the last part's identity differs from the original, phi statements in the successors are rewritten to source from the new last block. Edge data is preserved.

Returns `(b_pre, b_mid, b_post)` where `b_pre`/`b_post` are None when their statement list is empty. `mid_stmts` must be non-empty.

**Return type**
tuple[Block | None, Block, Block | None]

**Parameters**

- **graph** (*DiGraph*)
- **block** (*Block*)
- **pre_stmts** (*Sequence[Statement]*)
- **mid_stmts** (*Sequence[Statement]*)
- **post_stmts** (*Sequence[Statement]*)
- **block_addr_alloc** (*Callable[[], int]*)
- **idx_alloc** (*Callable[[], int]*)

### angr.analyses.decompiler.known_patterns.containing_record

The CONTAINING_RECORD macro (Windows drivers) / container_of (Linux kernel):

CONTAINING_RECORD(addr, type, field) == (type *)((char *)addr - offsetof(...))

which compiles to a bare pointer-minus-small-constant, `(p Sub off)`. Too generic to run unsolicited (any pointer arithmetic matches), so opt-in. The matched offset is captured as an extra call argument, and the return type is a pointer to a record struct reserving the leading `off` bytes (so Typehoon can grow it from the record's field accesses).

Arch-independent shape (the offset is a captured constant, not word-scaled).

Three qualifications keep the presentation honest, all measured against putty (user-space C, 139 real uses, DWARF `offsetof` as the oracle):

- One call per idiom. No compiler keeps the adjusted pointer around: it folds the subtraction into every field displacement that reads through it, so one source `container_of` reached the matcher as 3.4 separate matches on the same base, each naming a record base wrong by that field's offset (83.5% of the non-genuine matches were explained this way, against 13.0% for random offsets). `collapse_capture` keeps one match per base pointer; the largest offset is the right one to keep, since a field at member offset `m` of the record appears as `p - (off - m)` and so can only shrink the constant. Measured: this reports the true `container_of` offset for 80.1% (gcc) / 86.7% (clang) of base pointers, against 29% for the per-field matches.
- An offset floor. `p - 1` is a decrement, not a record base; offsets below one machine word carry no record. 29 of the 85 distinct offsets emitted in container_of-free code were exactly 1.

Rejected: requiring the result to be dereferenced. The captured base is a bare virtual variable of unknown type, so a decrement matches as happily as a record base, and demanding that the adjusted pointer sit inside a memory address looks like the way to tell them apart. It selects the wrong cases. The folded form is a field load and passes; the unfolded form, the one that survives because the code does something with the record other than read one field of it, which is the case the pattern gets right, does not, since the pointer is handed to a callee or stored. Measured on putty/gcc it dropped function-level recall from 90.7% to 65.0% and site-level recall from 42.7% to 2.1%, and it also rejects the `sum_list` fixture below. The offset floor already removes the decrements it was aimed at.

### angr.analyses.decompiler.known_patterns.context

PatternContext: the architecture / platform / language facts a KnownPattern template needs to instantiate itself for a given binary.

A pattern is defined once as a `KnownPatternTemplate` whose `build(ctx)` computes concrete field offsets, load sizes, and return types from a `PatternContext`, so a single definition covers 32- and 64-bit, libstdc++ and MSVC, etc.

angr.analyses.decompiler.known_patterns.context.**size_t_typename**(*bits*)

The C spelling of size_t for a given word width.

**Return type**
str

**Parameters**
**bits** (*int*)

angr.analyses.decompiler.known_patterns.context.**detect_linux_kernel_object**(*project*)

Whether the main object is a Linux kernel module or the kernel image.

Sections are checked first because they are the cheap and decisive signal: modpost emits `.modinfo` and `.gnu.linkonce.this_module` into every `.ko`, and they survive stripping. The `__ksymtab`/`__kcrctab` symbol scan additionally catches a vmlinux and modules whose section names were renamed.

**Return type**
bool

**Parameters**
**project** (Project)

angr.analyses.decompiler.known_patterns.context.**detect_windows_kernel_driver**(*project*)

Whether the main object is a Windows kernel-mode driver, i.e. imports the kernel executive / HAL / a kernel-mode port or class library instead of the user-mode Win32 DLLs.

**Return type**
bool

**Parameters**
**project** (Project)

angr.analyses.decompiler.known_patterns.context.**detect_cxx_runtime**(*project*)

Identify the C++ standard-library runtime: "msvc", "libstdcxx", or None. Mirrors the old is_cpp_binary / is_msvc_cpp_binary heuristics: dependency names first, then mangled-symbol evidence (mingw PEs link libstdc++ and use its layouts even though they are PEs), then, for statically linked, stripped binaries that carry neither, RTTI type descriptors embedded in data sections.

**Return type**
str | None

**Parameters**
**project** (Project)

**class** angr.analyses.decompiler.known_patterns.context.**PatternContext**

Bases: object

Target facts used to instantiate pattern templates.

The last two fields are binary evidence: facts about what kind of program this is, used by *gating* to switch whole pattern families on for targets where the idioms are certain to appear and off everywhere else.

**arch_name:** str

**bits:** int

**ptr_size:** int

**platform:** str | None

**cxx_runtime:** str | None

**is_cpp:** bool

**is_linux_kernel_object:** bool = False

**is_windows_kernel_driver:** bool = False

**classmethod from_project**(*project*)

**Return type**
*PatternContext*

**Parameters**
**project** (Project)

**word**(*n*)

Byte offset of the n-th pointer-word field.

**Return type**
int

**Parameters**
**n** (*int*)

**property word_size: int**

Load size (bytes) of a pointer / size_t field.

**property language: str**

**__init__**(*arch_name*, *bits*, *ptr_size*, *platform*, *cxx_runtime*, *is_cpp*, *is_linux_kernel_object=False*, *is_windows_kernel_driver=False*)

**Parameters**

- **arch_name** (*str*)
- **bits** (*int*)
- **ptr_size** (*int*)
- **platform** (*str | None*)
- **cxx_runtime** (*str | None*)
- **is_cpp** (*bool*)
- **is_linux_kernel_object** (*bool*)
- **is_windows_kernel_driver** (*bool*)

**Return type**
None

### angr.analyses.decompiler.known_patterns.ctype_tables

The glibc ctype macros: isspace, isdigit, … and tolower/toupper.

<ctype.h> classifies a character by indexing a locale table rather than by comparing ranges:

```
#define __isctype(c, type)  ((*__ctype_b_loc ())[(int) (c)] & (unsigned short int) type)
```

so `isspace(c)` compiles to a call to `__ctype_b_loc`, a load of the table pointer it returns, a 16-bit indexed load, and a mask. Every predicate is that one shape with a different constant, which is why they are generated here rather than written out.

Measured over the -O2 benchmark corpus, the eleven predicates account for 7,894 macro-oracle sites across 24 projects, wider project coverage than any family KnownPatterns covers today, with `tolower` and `toupper` adding 1,588 more.

The callee is what makes the whole family self-guarding: nothing else calls `__ctype_b_loc`. *PCallResult* binds the table to that call without consuming it, which matters because the compiler hoists the call out of loops and indexes the table from every predicate in the function.

Two byte lanes. glibc's `_ISbit(n)` is `n < 8 ? (1 << n) << 8 : (1 << n) >> 8`, so eight of the twelve masks live in the high byte of the table entry and four in the low byte, and gcc tests a byte rather than the 16-bit word. Hence three spellings per predicate rather than one: the mask as written against the whole entry, and, for a high mask, an 8-bit slice at byte 1 against the mask's high byte (`test $0x20,%ah`) or that byte loaded on its own (`testb $0x20, 1(%rax,%rdx,2)`); for a low mask, an 8-bit slice at byte 0. The slice spellings are not interchangeable with the first: structural matching skips a Convert and does not skip an Extract.

Calibrated against tests/x86_64/decompiler/known_patterns_ctype (gcc 12.2.0 -O2).

angr.analyses.decompiler.known_patterns.ctype_tables.**CTYPE_B_LOC = frozenset({'___ctype_b_loc', '__ctype_b_loc'})**

glibc's `__ctype_b_loc` under every spelling a binary may know it by.

angr.analyses.decompiler.known_patterns.ctype_tables.**CTYPE_PREDICATES: tuple[tuple[str, int], ...] = (('isupper', 256), ('islower', 512), ('isalpha', 1024), ('isdigit', 2048), ('isxdigit', 4096), ('isspace', 8192), ('isprint', 16384), ('isgraph', 32768), ('isblank', 1), ('iscntrl', 2), ('ispunct', 4), ('isalnum', 8))**

`(macro name, mask)`. The values are glibc's `enum` from <ctype.h>, computed by `_ISbit(n) = n < 8 ? (1 << n) << 8 : (1 << n) >> 8`.

### angr.analyses.decompiler.known_patterns.dsl

Declarative pattern-AST for describing known AIL code idioms (KnownPatterns).

Pattern nodes are frozen dataclasses that structurally match AIL expressions and statements. Every node accepts an optional `name`: a named *capture*. When the same capture name occurs more than once in a pattern, all occurrences must bind `.likes()`-equal expressions (unification).

Matching is performed by calling `PatternNode.match()` with a *MatchState* (the bindings environment) and a *MatchCtx* (matcher-wide flags and callbacks). Match methods are functional: they return a new `MatchState` on success and `None` on failure, so alternatives (commutative operand orders, *PChoice*) can backtrack by reusing the pre-branch state.

**class** angr.analyses.decompiler.known_patterns.dsl.**MatchState**

Bases: `object`

The bindings environment threaded through a match attempt.

**bindings:** `dict[str, Expression]`

**consumed_stmt_idxs:** `frozenset[int] = frozenset({})`

**chased_defs:** `tuple[tuple[int, int], ...] = ()`

**remote_defs:** `tuple[tuple[int, Expression], ...] = ()`

**skipped_converts:** `tuple[Convert, ...] = ()`

**base_aliases:** `tuple[tuple[int, int, int], ...] = ()`

**bind**(*name*, *expr*)

Bind `name` to `expr`; on rebind, require `.likes()` equality.

**Return type**
*MatchState* | None

**Parameters**

- **name** (*str*)
- **expr** (Expression)

**__init__**(*bindings=<factory>*, *consumed_stmt_idxs=frozenset({})*, *chased_defs=()*, *remote_defs=()*, *skipped_converts=()*, *base_aliases=()*)

**Parameters**

- **bindings** (*dict[str, Expression]*)
- **consumed_stmt_idxs** (*frozenset[int]*)

- **chased_defs** (*tuple[tuple[int, int], ...]*)
- **remote_defs** (*tuple[tuple[int, Expression], ...]*)
- **skipped_converts** (*tuple[Convert, ...]*)
- **base_aliases** (*tuple[tuple[int, int, int], ...]*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**MatchCtx**

Bases: object

Matcher-wide flags and callbacks.

**ptr_bits**: int | None = None

**skip_conversions**: bool = True

**chase_fn**: Callable[[int], tuple[int, *Expression*] | None] | None = None

**peek_fn**: Callable[[int], *Expression* | None] | None = None

**remote_chase_fn**: Callable[[int], *Expression* | None] | None = None

**stack_slot_fn**: Callable[[int], *VirtualVariable* | None] | None = None

**call_target_fn**: Callable[[*Call*], frozenset[str]] | None = None

**next_idx**()

**Return type**
int

**def_fn**: Callable[[int], *Expression* | None] | None = None

**__init__**(*ptr_bits=None*, *skip_conversions=True*, *chase_fn=None*, *peek_fn=None*, *remote_chase_fn=None*, *stack_slot_fn=None*, *call_target_fn=None*, *next_idx=<function _standalone_idx>*, *def_fn=None*)

**Parameters**

- **ptr_bits** (*int | None*)
- **skip_conversions** (*bool*)
- **chase_fn** (*Callable[[int], tuple[int, Expression] | None] | None*)
- **peek_fn** (*Callable[[int], Expression | None] | None*)
- **remote_chase_fn** (*Callable[[int], Expression | None] | None*)
- **stack_slot_fn** (*Callable[[int], VirtualVariable | None] | None*)
- **call_target_fn** (*Callable[[Call], frozenset[str]] | None*)
- **next_idx** (*Callable[[], int]*)
- **def_fn** (*Callable[[int], Expression | None] | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PatternNode**

Bases: object

Base class of all pattern nodes.

class angr.analyses.decompiler.known_patterns.dsl.**PatternExpr**

Bases: *PatternNode*

Base class of expression patterns.

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

class angr.analyses.decompiler.known_patterns.dsl.**PAny**

Bases: *PatternExpr*

Matches any expression.

**name:** str | None = None

**bits:** int | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*name=None*, *bits=None*)

**Parameters**

- **name** (*str | None*)
- **bits** (*int | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PVVar**

Bases: *PatternExpr*

Matches a VirtualVariable.

**name:** str | None = None

**bits:** int | None = None

**categories:** frozenset[VirtualVariableCategory] | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
MatchState | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*name=None*, *bits=None*, *categories=None*)

**Parameters**

- **name** (*str | None*)
- **bits** (*int | None*)
- **categories** (*frozenset[VirtualVariableCategory] | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.dsl.**PConst**

Bases: *PatternExpr*

Matches a Const, by exact value or by predicate. Width is ignored unless `bits` is given (e.g. shift amounts are often 8-bit).

**value:** int | None = None

**pred:** Callable[[int], bool] | None = None

**bits:** int | None = None

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
MatchState | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*value=None*, *pred=None*, *bits=None*, *name=None*)

**Parameters**

- **value** (*int | None*)
- **pred** (*Callable[[int], bool] | None*)
- **bits** (*int | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PBinOp**

Bases: *PatternExpr*

Matches a BinaryOp. op may be a single op string or a set of acceptable ops. Commutative ops try both operand orders unless commutative is explicitly False.

**op:** str | frozenset[str]

**operands:** tuple[*PatternExpr*, *PatternExpr*]

**commutative:** bool | None = None

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*op*, *operands*, *commutative=None*, *name=None*)

**Parameters**

- **op** (*str | frozenset[str]*)
- **operands** (*tuple[PatternExpr, PatternExpr]*)
- **commutative** (*bool | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PUnaryOp**

Bases: *PatternExpr*

Matches a UnaryOp.

**op:** str | frozenset[str]

**operand:** *PatternExpr*

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)

- **ctx** (MatchCtx)

**__init__**(*op*, *operand*, *name=None*)

**Parameters**

- **op** (*str | frozenset[str]*)
- **operand** (PatternExpr)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PConv**

Bases: *PatternExpr*

Matches a Convert explicitly (never skipped).

**operand:** *PatternExpr*

**from_bits:** int | None = None

**to_bits:** int | None = None

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*operand*, *from_bits=None*, *to_bits=None*, *name=None*)

**Parameters**

- **operand** (PatternExpr)
- **from_bits** (*int | None*)
- **to_bits** (*int | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PExtract**

Bases: *PatternExpr*

Matches an `Extract`, a bit-slice of a wider value.

This is how a scalar reaches integer code out of a vector register when the 128-bit op did not get narrowed away: `movq %xmm2,%rax` on a value that came from `maxsd`/`mulsd` lifts to `Extract(vvar_128, 64bits@0)` rather than to a Convert. Without a node for it, every libm bit-twiddle whose operand is a computed double rather than an incoming argument is unmatchable.

`bits`/`offset` default to unconstrained; `offset` is in bits.

operand: *PatternExpr*

bits: int | None = None

offset: int | None = None

name: str | None = None

match(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

__init__(*operand*, *bits=None*, *offset=None*, *name=None*)

**Parameters**

- **operand** (PatternExpr)
- **bits** (*int | None*)
- **offset** (*int | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PLoad**

Bases: *PatternExpr*

Matches a memory Load. `size` is in bytes.

addr: *PatternExpr*

size: int | None = None

name: str | None = None

match(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

__init__(*addr*, *size=None*, *name=None*)

**Parameters**

- **addr** (PatternExpr)
- **size** (*int | None*)

- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.dsl.**PCall**

Bases: *PatternExpr*

Matches a `Call` whose callee is one of `names`.

The callee is what makes a pattern containing this node self-guarding: an idiom spelled around a named runtime function (`operator delete` in the std::string destructor, `__ctype_b_loc` in `isspace`) cannot be confused with arithmetic that happens to look alike, because no other code calls that function to do something else.

`names` is matched against every spelling the callee is known by (the raw symbol and its demangled form), so a pattern may name either. Prefer the mangled spelling: it is exact, while a demangled one carries the argument list and varies with the demangler.

`args=None` leaves the arguments unconstrained; a tuple constrains them positionally, with `None` in a slot meaning "any expression".

**names:** `frozenset[str]`

**args:** `tuple[PatternExpr | None, ...] | None = None`

**name:** `str | None = None`

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*names*, *args=None*, *name=None*)

**Parameters**

- **names** (*frozenset[str]*)
- **args** (*tuple[PatternExpr | None, ...] | None*)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.dsl.**PCallResult**

Bases: *PatternExpr*

The value a call to one of `names` returned.

A call's result is rarely used where it is produced: the compiler assigns it to a register and reads that register, often in another block. `__ctype_b_loc()` is called once per function and its table indexed at every predicate. So this node matches the virtual variable and looks up its definition read-only. Nothing is consumed, moved or recomputed; the call stays where the compiler put it.

An inline `Call` expression is matched too, for the case where the value is used at its definition.

**names:** frozenset[str]

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
MatchState | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*names*, *name=None*)

**Parameters**

- **names** (*frozenset[str]*)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.dsl.**PDefOf**

Bases: *PatternExpr*

`inner`, or a virtual variable whose reaching definition is `inner`.

The compiler is free to compute a value once and use it twice, and it does so exactly where an idiom uses one value twice. `if (_M_p != &_M_local_buf) operator delete(_M_p, ...)` reads _M_p in the compare and in the call, so gcc loads it into a register and the pattern sees `Load(s)` in neither place; it sees the same virtual variable in both.

Reading that variable's definition is read-only. Nothing is consumed, moved or recomputed, and the identification stays accurate even if memory changes afterwards: the variable holds what the load produced at the definition, which is what the idiom meant. That is why this is not `peek_fn`, whose contract ("can be recomputed at the use") a load cannot satisfy.

The unconsumed definition is left where it was, so a region that matches through this node keeps it as residue. That is harmless, and dead-code elimination removes it once the idiom's other uses are gone.

**inner:** *PatternExpr*

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
MatchState | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*inner*, *name=None*)

**Parameters**

- **inner** (PatternExpr)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PField**

Bases: *PatternExpr*

The *address* of the field at `offset` inside the object bound to `base`.

Spelling a container's fields as `PLoad(PVVar("v"))` / `PLoad(PVVar("v") + 8)` requires the object to sit at the very address held in a virtual variable. That is the minority case: measured over the benchmark corpus, only 10-25% of the DWARF sites for a `std::vector` accessor read the container through a bare pointer; the rest reach it as a *field of something else* (`this->tokens.size()`), which lifts to `Load(this + 56)` / `Load(this + 64)` and cannot bind `v` at all.

`PField` matches such an address and binds `base` to the object's own address: `root` when the object is at offset 0, else `root + K`. Two fields of one object therefore unify on `K`, which is what keeps the node honest: it replaces the constraint "this displacement is exactly 8" with "these two displacements differ by exactly 8".

Only use it in patterns that reference at least two fields. With a single field there is no second displacement to constrain, so `PLoad(PField("s", 8))` degenerates into "any load through a pointer plus a non-negative constant" and matches essentially everything.

`root` must be a virtual variable: the outliner materializes `tmp = root + K` ahead of the region so the synthesized call still takes a variable.

**base:** *str*

**offset:** *int*

**name:** *str | None* = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*base*, *offset*, *name=None*)

**Parameters**

- **base** (*str*)
- **offset** (*int*)
- **name** (*str | None*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.dsl.**PStackField**

Bases: *PatternExpr*

A field of a container that lives on the stack.

A stack object never reaches the matcher as an object at all: variable recovery has already split it into one virtual variable per slot, so `std::string s;` is a handful of independent `vvar{s-112}`, `vvar{s-96}` and there is no `Load(s)` to match and no `s + 16` to compare against. Measured over the benchmark corpus (*kp-eval/stack_share.py*), stack objects are 40-54% of the DWARF sites for `std::string::length` and 15-35% for `std::vector<T>::size`.

This node matches the slot of the field at `offset` and binds `base` to the object's own stack offset, so two fields of one object unify on it exactly as *PField*'s two displacements do. `as_address` matches the address form (`&slot`, which lifts to `UnaryOp("Reference", vvar)`) rather than the value form; `std::string::capacity` needs both, since it compares the data pointer against the address of the local buffer.

The binding is a `StackBaseOffset`, the object's own address. That is both the thing two fields must agree on and, directly, the argument the synthesized call takes, so nothing has to be materialized for it.

Matching is only half the problem, and the other half is why a pattern using this node is never outlined: the region for a stack container reads N independent slots, so its live-ins are N values rather than one pointer and there is nothing for a synthesized callee to take. Such a match is instead rewritten in place: the matched expression, or the whole region, is replaced by `call(&s)`, whose argument the binding already is. See `KnownPatternFinder._rewrite_in_place` and `_rewrite_graph_in_place`.

**base:** *str*

**offset:** *int*

**as_address:** *bool* = **False**

**size:** *int* | *None* = **None**

**name:** *str* | *None* = **None**

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*base*, *offset*, *as_address=False*, *size=None*, *name=None*)

**Parameters**

- **base** (*str*)
- **offset** (*int*)
- **as_address** (*bool*)
- **size** (*int | None*)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.PPhi

Bases: *PatternExpr*

Matches a Phi expression, capturing it whole (its individual sources are not matched). Useful for loop-header blocks, whose statements are phi assignments.

**name:** str | None = None

**bits:** int | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*name=None*, *bits=None*)

**Parameters**

- **name** (*str | None*)
- **bits** (*int | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.PITE

Bases: *PatternExpr*

Matches an ITE (ternary) expression `cond ? iftrue : iffalse`.

Needed because ITERegionConverter (AFTER_GLOBAL_SIMPLIFICATION) collapses `if (c) x = a; else x = b;` diamonds into a single ITE assignment before the KnownPatternOutliner runs, so a two-armed value-select idiom reaches the outliner as an expression, not as a PGraphPat region. (A triangle, with one arm empty as in the MSVC std::string::c_str SSO select, is not converted, which is why that idiom is still a PGraphPat.)

**cond:** *PatternExpr*

**iftrue:** *PatternExpr*

**iffalse:** *PatternExpr*

**name:** str | None = None

**match**(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

__init__(*cond*, *iftrue*, *iffalse*, *name=None*)

**Parameters**

- **cond** (PatternExpr)
- **iftrue** (PatternExpr)
- **iffalse** (PatternExpr)
- **name** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PChoice**

Bases: *PatternExpr*

Ordered alternation: matches the first alternative that succeeds.

**alternatives:** tuple[*PatternExpr*, ...]

__init__(**alternatives*)

**Parameters**
**alternatives** (PatternExpr)

match(*expr*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **expr** (Expression)
- **state** (MatchState)
- **ctx** (MatchCtx)

class angr.analyses.decompiler.known_patterns.dsl.**PatternStmt**

Bases: *PatternNode*

Base class of statement patterns.

match(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

class angr.analyses.decompiler.known_patterns.dsl.**PAssign**

Bases: *PatternStmt*

Matches an Assignment statement.

**dst:** *PatternExpr*

**src:** *PatternExpr*

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*dst*, *src*)

**Parameters**

- **dst** (PatternExpr)
- **src** (PatternExpr)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.dsl.**PStore**

Bases: *PatternStmt*

Matches a Store statement. `size` is in bytes.

**addr:** *PatternExpr*

**value:** *PatternExpr*

**size:** int | None = None

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*addr*, *value*, *size=None*)

**Parameters**

- **addr** (PatternExpr)
- **value** (PatternExpr)
- **size** (*int | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PCallStmt**

Bases: *PatternStmt*

Matches a statement whose effect is a call.

Ailment has no Call statement: a call whose result is discarded is a `SideEffectStatement` wrapping the Call expression, and one whose result is kept is an ordinary `Assignment`. Both spellings are the same idiom, so both match; `dst`, when given, also constrains the assigned destination, which requires the assignment form.

**call:** *PCall*

**dst:** *PatternExpr* | None = None

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*call*, *dst=None*)

**Parameters**

- **call** (PCall)
- **dst** (PatternExpr | *None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PCondJump**

Bases: *PatternStmt*

Matches a ConditionalJump, matching its condition expression.

**condition:** *PatternExpr*

**match**(*stmt*, *state*, *ctx*)

**Return type**
*MatchState* | None

**Parameters**

- **stmt** (Statement)
- **state** (MatchState)
- **ctx** (MatchCtx)

**__init__**(*condition*)

**Parameters**
**condition** (PatternExpr)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PStmtSeq**

Bases: *PatternStmt*

Matches a group of statements within one block. When `ordered` (the default), the statement patterns must match in order; with `allow_gaps`, unrelated statements may sit between the matched ones. When `ordered` is False, the statement patterns match statements of the block in any order (each pattern to a distinct statement), which suits idioms whose statement order the compiler chooses freely (e.g. list-link stores).

**stmts:** `tuple[PatternStmt, ...]`

**allow_gaps:** `bool = True`

**ordered:** `bool = True`

**max_gap:** `int = 8`

How many unmatched statements may sit between two matched ones. Without a bound the scan runs to the end of the block from every statement it could start at, quadratic per block per template, and finds nothing: an idiom's statements are adjacent apart from what the scheduler interleaved, which is a handful of instructions rather than a basic block.

**__init__**(*stmts*, *allow_gaps=True*, *ordered=True*, *max_gap=8*)

**Parameters**

- **stmts** (*tuple[PatternStmt, ...]*)
- **allow_gaps** (*bool*)
- **ordered** (*bool*)
- **max_gap** (*int*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PBlockPat**

Bases: *PatternNode*

A labeled block of statements in a multi-block pattern graph. An empty `stmts` sequence matches any block (structure alone identifies it).

**label:** `str`

**stmts:** *PStmtSeq*

**__init__**(*label*, *stmts*)

**Parameters**

- **label** (*str*)
- **stmts** (PStmtSeq)

**Return type**
None

class angr.analyses.decompiler.known_patterns.dsl.**PGraphPat**

Bases: *PatternNode*

A multi-block pattern: labeled blocks connected by edges.

`blocks` maps a label to its *PBlockPat*. `edges` are `(src_label, dst_label)` pairs; a destination label that does not appear in `blocks` denotes an external successor, a region exit that becomes the Outliner frontier. `entry` is the label of the single entry block. Captures unify across all blocks (one bindings environment).

**blocks:** dict[str, *PBlockPat*]

**edges:** Sequence[tuple[str, str]]

**entry:** str

**property external_labels:** set[str]

**__init__**(*blocks*, *edges*, *entry*)

**Parameters**

- **blocks** (*dict[str, PBlockPat]*)
- **edges** (*Sequence[tuple[str, str]]*)
- **entry** (*str*)

**Return type**
None

angr.analyses.decompiler.known_patterns.dsl.**pattern_anchor_key**(*node*)

A cheap discriminator of the pattern root, used to prune candidate patterns at each expression position: `(ail kind, op or None)`.

Returns None when the root is not discriminating (PAny/PChoice/...), in which case the pattern must be attempted everywhere.

**Return type**
tuple[str, str | None] | None

**Parameters**
**node** (PatternNode)

angr.analyses.decompiler.known_patterns.dsl.**stmt_pattern_anchor_key**(*node*)

The discriminator of a *statement* pattern's first statement.

Statement patterns are anchored at every statement of every block and then scan forward, so trying one that cannot possibly match is the expensive kind of waste. The key is the statement kind plus the anchor key of the expression that carries the pattern's shape (an assignment's source, a store's value), which is what separates `t = *a` (a swap) from the assignments that make up most of a block.

**Return type**
tuple[str, tuple[str, str | None] | None] | None

**Parameters**
**node** (PatternNode)

angr.analyses.decompiler.known_patterns.dsl.**stmt_anchor_key**(*stmt*)

The anchor key of a concrete AIL statement, for pruning.

**Return type**
tuple[str, tuple[str, str | None] | None]

**Parameters**
**stmt** (Statement)

angr.analyses.decompiler.known_patterns.dsl.**expr_anchor_key**(*expr*)

The anchor key of a concrete AIL expression, for pruning.

**Return type**
tuple[str, str | None]

**Parameters**
**expr** (Expression)

### angr.analyses.decompiler.known_patterns.finder

KnownPatternFinder: find occurrences of KnownPatterns in a Clinic AIL graph and outline them into calls via the Outliner analysis.

**exception** angr.analyses.decompiler.known_patterns.finder.**UnsupportedOutlineError**

Bases: Exception

Raised when a match cannot be safely outlined (e.g. side-effecting statements interleave with the matched span, or the recovered callee interface does not agree with the pattern's declared parameters).

**class** angr.analyses.decompiler.known_patterns.finder.**KnownPatternMatch**

Bases: object

One occurrence of a KnownPattern in an AIL graph.

For expression-level matches, the boundary is `consumed_stmt_idxs` (statements wholly owned by the match, e.g. chased definitions) plus `expr_path` (the location of the matched sub-expression inside the anchor statement). For statement-sequence matches, `stmt_span` holds the ordered matched statement indices (the anchor is the first) and `matched_expr` is None.

`duplicated_stmt_idxs` holds chased definitions that are copied into the outlined region instead of moved: the originals stay where they are (other code still uses them) and the copies make the pattern's captured values the region's live-ins. It is always disjoint from `consumed_stmt_idxs`.

`recomputed_defs` does the same for definitions that live in *another* block (a CSE'd value, possibly reaching the anchor through a phi): the expression is pure, so the region recomputes it from the captured operands.

**pattern:** *KnownPattern*

**block_loc:** tuple[int, int | None]

**anchor_stmt_idx:** int

**expr_path:** tuple[tuple[str, int | None], ...]

**consumed_stmt_idxs:** frozenset[int]

**captures:** dict[str, *Expression*]

**matched_expr:** *Expression* | None

**duplicated_stmt_idxs:** frozenset[int] = frozenset({})

**recomputed_defs:** tuple[tuple[int, *Expression*], ...] = ()

**base_aliases:** tuple[tuple[int, int, int], ...] = ()

**stmt_span:** tuple[int, ...] | None = None

**block_map:** dict[str, tuple[int, int | None]] | None = None

**consumed_by_block:** dict[tuple[int, int | None], frozenset[int]] | None = None

**frontier_locs:** frozenset[tuple[int, int | None]] | None = None

__init__(*pattern*, *block_loc*, *anchor_stmt_idx*, *expr_path*, *consumed_stmt_idxs*, *captures*, *matched_expr*, *duplicated_stmt_idxs=frozenset({})*, *recomputed_defs=()*, *base_aliases=()*, *stmt_span=None*, *block_map=None*, *consumed_by_block=None*, *frontier_locs=None*)

**Parameters**

- **pattern** (KnownPattern)
- **block_loc** (*tuple[int, int | None]*)
- **anchor_stmt_idx** (*int*)
- **expr_path** (*tuple[tuple[str, int | None], ...]*)
- **consumed_stmt_idxs** (*frozenset[int]*)
- **captures** (*dict[str, Expression]*)
- **matched_expr** (*Expression | None*)
- **duplicated_stmt_idxs** (*frozenset[int]*)
- **recomputed_defs** (*tuple[tuple[int, Expression], ...]*)
- **base_aliases** (*tuple[tuple[int, int, int], ...]*)
- **stmt_span** (*tuple[int, ...] | None*)
- **block_map** (*dict[str, tuple[int, int | None]] | None*)
- **consumed_by_block** (*dict[tuple[int, int | None], frozenset[int]] | None*)
- **frontier_locs** (*frozenset[tuple[int, int | None]] | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.finder.**OutlineResult**

Bases: object

The outcome of outlining one KnownPatternMatch.

**graph:** DiGraph

**match:** *KnownPatternMatch*

**call_stmt:** *Statement*

**child_func:** *Function* | None

**child_graph:** DiGraph | None

**child_funcargs:** list[*VirtualVariable*]

__init__(*graph*, *match*, *call_stmt*, *child_func*, *child_graph*, *child_funcargs*)

**Parameters**

- **graph** (*DiGraph*)
- **match** (KnownPatternMatch)
- **call_stmt** (Statement)
- **child_func** (Function | *None*)

- **child_graph** (*DiGraph | None*)
- **child_funcargs** (*list[VirtualVariable]*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.finder.**KnownPatternFinder**

Bases: *Analysis*

Finds occurrences of KnownPatterns in a Clinic AIL graph.

The graph to match on is the final (SSA-form) Clinic graph, i.e. `decompiler.ail_graph` / `clinic.cc_graph`. Matches are reported in `self.matches`; *outline()* turns a match into a synthesized call by invoking the Outliner analysis on a copy of the graph.

**MAX_REMOTE_CHASE_DEPTH = 6**

**__init__**(*func*, *ail_graph*, *patterns=None*, *chase_defs=True*, *chase_remote_defs=True*, *skip_conversions=True*, *vvar_id_start=48879*, *block_addr_start=2864381952*, *ail_manager=None*, *force_patterns=None*)

`patterns` may be pattern templates (KnownPatternTemplate) and/or already-concrete KnownPatterns; templates are instantiated for this binary's PatternContext. When None, the registry is used, filtered down to the templates that are enabled for this target: default-on ones, ones named by `force_patterns`, and ones whose gate opens (see *gating*).

`force_patterns` is the user's force-enable selection, `"all"` or an iterable of template names / call names, and is only consulted when `patterns` is None (an explicit pattern list is already a selection).

**Parameters**

- **func** (Function)
- **ail_graph** (*DiGraph*)
- **patterns** (*Iterable[object] | None*)
- **chase_defs** (*bool*)
- **chase_remote_defs** (*bool*)
- **skip_conversions** (*bool*)
- **vvar_id_start** (*int*)
- **block_addr_start** (*int*)
- **force_patterns** (*str | Iterable[str] | None*)

**matches: list[*KnownPatternMatch*]**

**call_target_names**(*call*)

Public face of `_resolve_call_target()`, for the outliner's clean-up of calls a pattern declared pure.

**Return type**
frozenset[str]

**Parameters**
**call** (Call)

**apply_matches**(*matches*, *ail_graph*)

Apply as many of `matches` as possible, batching where it matters.

Applying one match at a time is fine until several of them interleave in one block, which is exactly what consecutive field swaps do:

**t1 = x->b; t2 = y->b; y->b = t1; x->b = t2;**
^ the next field's load sits in this one's gap

Rewrite the first and the second's statements have moved; rewrite it anyway and the third now has an opaque call sitting in its gap, which no disjointness test will let it move across. So the void statement-span matches of a block, which the finder has already checked do not conflict, are spliced in together, against the statement indices they were found at. Everything else keeps the one-at-a-time path.

Returns the new graph and the ids of the matches that were applied.

**Return type**
tuple[DiGraph, set[int]]

**Parameters**

- **matches** (*list[KnownPatternMatch]*)
- **ail_graph** (*DiGraph*)

**outline**(*match*, *ail_graph=None*)

Outline one match: split the anchor block so the matched span forms its own block, invoke the Outliner analysis on it, and rewrite the synthesized callsite into the pattern's call. Operates on (and returns) a copy; neither `self._graph` nor `ail_graph` is mutated.

**Return type**
*OutlineResult*

**Parameters**

- **match** (KnownPatternMatch)
- **ail_graph** (*DiGraph | None*)

**MAX_LIFTED_EXPR_NODES = 8**

Node budget for a lifted parameter expression; see _is_liftable.

**outline_at**(*location*, *pattern*, *ail_graph*)

Find `pattern` at `location` (`(block_addr, block_idx)` or `(block_addr, block_idx, stmt_idx)`) in `ail_graph` and outline it.

**Return type**
*OutlineResult*

**Parameters**

- **location** (*tuple[int, int | None] | tuple[int, int | None, int]*)
- **pattern** (KnownPattern)
- **ail_graph** (*DiGraph*)

### angr.analyses.decompiler.known_patterns.gating

Criteria gates: per-target (and per-function) enablement for pattern templates.

A *KnownPatternTemplate* carries static applicability filters (arch / language / runtime / platform) and a boolean `default_enabled`. Many idioms are opt-in because their shape is too generic to ship on every binary: `Load(p) == 0` is a list-emptiness test in a kernel module and a null check everywhere else.

A gate is the third channel: it lets an opt-in template say "turn me on when this evidence is present". Two kinds of evidence are modelled:

- target evidence (*TargetGate*): a fact about the binary as a whole, derived once into the *PatternContext*: this is a Linux kernel module, this is a Windows kernel-mode driver. Cheap, exact, and evaluated before any matching happens.
- corroboration (*CorroboratedBy*): another, unambiguous pattern already matched in the same function. This is what converts "too generic to ship" into "safe here": `Load(Load(s))` is meaningless on its own, but in a function where `std::string::length` (a load at the string's size offset) or `std::string::capacity` (the self-referential SSO select) already matched, the object really is a `std::string` and the load really is `front()`.

Corroboration needs the match results, so it is resolved by *KnownPatternFinder* in a second matching stage: the finder matches with the ungated set, collects the evidence, re-evaluates the gates that asked for it, and, only if that opens something, matches again with the enlarged set. Gates that do not need evidence are resolved up front.

A gate can only open an opt-in template; it never disables a default-on one.

**class** angr.analyses.decompiler.known_patterns.gating.**GateContext**

Bases: object

Everything a gate may look at.

**Variables**

- **ctx** – The target's *PatternContext* (arch, platform, C++ runtime, and the binary-evidence facts).
- **project** – The Project, for gates that need to look past the context.
- **evidence** – Names and call names of the patterns established in the function currently being matched: both the patterns that matched in the first stage and the known-pattern calls already present in the graph (outlined by an earlier round). Empty when gates are evaluated outside a function.

**ctx:** *PatternContext*

**project:** *Project* | None = None

**evidence:** frozenset[str] = frozenset({})

**with_evidence**(*evidence*)

**Return type**
*GateContext*

**Parameters**
**evidence** (*Iterable[str]*)

**__init__**(*ctx*, *project=None*, *evidence=frozenset({})*)

**Parameters**

- **ctx** (PatternContext)
- **project** (Project | *None*)
- **evidence** (*frozenset[str]*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.gating.**PatternGate**

Bases: object

Base class for template gates. Instances are callables so that a gate can be used anywhere a `Callable[[GateContext], bool]` is expected.

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence: bool**

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label: str**

**__init__**()

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.gating.**TargetGate**

Bases: *PatternGate*

Opens when a predicate over the target's PatternContext holds.

The predicate reads facts that were derived once from the loaded binary, so evaluating it is free and its result is the same for every function.

**name: str**

**predicate: Callable[[*PatternContext*], bool]**

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property label: str**

**__init__**(*name*, *predicate*)

**Parameters**

- **name** (*str*)
- **predicate** (*Callable[[PatternContext], bool]*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.gating.**CorroboratedBy**

Bases: *PatternGate*

Opens in a function where at least one of witnesses already matched.

Witnesses are given as template ``name``s or ``call_name``s; the finder puts both spellings of every established pattern into the evidence set.

**witnesses: frozenset[str]**

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence: bool**

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label: str**

**__init__**(*witnesses*)

**Parameters**
**witnesses** (*frozenset[str]*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.gating.**AnyOf**

Bases: *PatternGate*

Opens when any sub-gate opens.

**gates: tuple[*PatternGate*, ...]**

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence: bool**

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label: str**

**__init__**(*gates=<factory>*)

**Parameters**
**gates** (*tuple[PatternGate, ...]*)

**Return type**
None

**class** angr.analyses.decompiler.known_patterns.gating.**AllOf**

Bases: *PatternGate*

Opens when every sub-gate opens.

**gates: tuple[*PatternGate*, ...]**

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence: bool**

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label: str**

**__init__**(*gates=<factory>*)

**Parameters**
**gates** (*tuple[PatternGate, ...]*)

**Return type**
None

angr.analyses.decompiler.known_patterns.gating.**corroborated_by**(**witnesses*)

A gate that opens in a function where one of `witnesses` matched.

**Return type**
*CorroboratedBy*

**Parameters**
**witnesses** (*str*)

**class** angr.analyses.decompiler.known_patterns.gating.**CorroboratedByPattern**

Bases: *PatternGate*

Opens where evidence matching a *regex* was established.

`CorroboratedBy` takes a fixed list of witness names, which no longer scales: families are generated per element size, so "any std::vector<T>::size" is five hundred names. This matches the evidence set instead.

**regex: str**

**opens**(*gctx*)

**Return type**
bool

**Parameters**
**gctx** (GateContext)

**property requires_evidence: bool**

Whether this gate can only be decided once the function's matches are known. Such gates are evaluated a second time by the finder.

**property label: str**

**__init__**(*regex*)

**Parameters**
**regex** (*str*)

**Return type**
None

angr.analyses.decompiler.known_patterns.gating.**corroborated_by_pattern**(*regex*)

A gate that opens in a function where evidence matching `regex` was established.

**Return type**
*CorroboratedByPattern*

**Parameters**
**regex** (*str*)

angr.analyses.decompiler.known_patterns.gating.**any_of**(**gates*)

**Return type**
*AnyOf*

**Parameters**
**gates** (PatternGate)

angr.analyses.decompiler.known_patterns.gating.**all_of**(**gates*)

**Return type**
*AllOf*

**Parameters**
**gates** (PatternGate)

### angr.analyses.decompiler.known_patterns.generator

PatternGenerator: build a KnownPattern from a decompilation text selection.

This is the inverse of KnownPatternFinder. Given a range of the generated C source (Decompiler.codegen) and a target call, it resolves the selection back to AIL statements/expressions and emits the corresponding DSL pattern:

- an expression selection (a sub-expression of one statement) becomes a *PatternExpr* tree;
- a straight-line statement selection (statements of one basic block) becomes a *PStmtSeq*;
- a selection spanning several basic blocks becomes a *PGraphPat* (requires the finder's ail_graph for the block/edge structure).

Text offsets are resolved through the codegen position maps (map_pos_to_node / ailexpr2cnode / cnode2ailexpr); the rendered AIL objects are .likes()-equal to the finder's graph objects, so a generated pattern re-matches via KnownPatternFinder on the same ail_graph (the round-trip is the acceptance test).

**exception** angr.analyses.decompiler.known_patterns.generator.**PatternGenerationError**

Bases: Exception

Raised when a selection cannot be turned into a KnownPattern (ambiguous, unsupported, or inconsistent with the given arguments).

**class** angr.analyses.decompiler.known_patterns.generator.**PatternGenerator**

Bases: object

Generate a KnownPattern from a text selection in codegen.text.

**Parameters**

- **codegen** – the Decompiler.codegen (a CStructuredCodeGenerator).
- **ail_graph** (DiGraph | None) – the graph the generated pattern will be matched against (Decompiler.ail_graph). Required for multi-block (control-flow-spanning) selections; optional otherwise.

**__init__**(*codegen*, *ail_graph=None*)

**Parameters**
**ail_graph** (*DiGraph | None*)

**vvar_id_at**(*offset*)

The AIL vvar id of the variable rendered at `offset`, or None.

**Return type**
int | None

**Parameters**
**offset** (*int*)

**offset_of**(*needle*, *start=0*)

Convenience: the offset of `needle` in the rendered text.

**Return type**
int

**Parameters**

- **needle** (*str*)
- **start** (*int*)

**generate**(*start_offset*, *end_offset*, *call_name*, *arg_offsets*, *, *name=None*, *display_name=None*, *returnty=None*, *param_types=None*, *const_arg_offsets=()*, *arches=None*, *platforms=None*, *binary_guard=None*, *default_enabled=False*)

**Return type**
*KnownPattern*

**Parameters**

- **start_offset** (*int*)
- **end_offset** (*int*)
- **call_name** (*str*)
- **arg_offsets** (*Sequence[int]*)
- **name** (*str | None*)
- **display_name** (*str | None*)
- **returnty** (*str | CppRef | None*)
- **param_types** (*Sequence[str | CppRef | None] | None*)
- **const_arg_offsets** (*Sequence[int]*)
- **arches** (*tuple[str, ...] | None*)
- **platforms** (*tuple[str, ...] | None*)
- **default_enabled** (*bool*)

### angr.analyses.decompiler.known_patterns.kernel_err

Linux kernel <linux/err.h> error-pointer idioms.

`IS_ERR(p)` tests `(unsigned long)p >= (unsigned long)-MAX_ERRNO`; the threshold is derived from the context's word width, so one definition covers 32- and 64-bit.

gcc emits the test in four different shapes depending on where it lands, and a pattern that covers only one of them silently misses most real code: the literal `>= THR`, the strength-reduced `> THR-1` (the value form at -O2), and the operand-swapped `<=` / `<` forms (the branch-condition form). All four are matched. The comparison must be unsigned

(a signed compare against the same constant is an unrelated test), which is checked in `where` because the DSL does not model `BinaryOp.signed`.

`PTR_ERR` and `ERR_PTR` are pure casts that compile to a bare register move and leave no residue at all, so there is nothing to match and they are deliberately absent.

Opt-in: these are kernel idioms, and renaming a comparison to `IS_ERR()` in a user-space binary would be misleading even though the threshold constant is in practice unique to err.h. Calibrated against tests/x86_64/decompiler/known_patterns_kernel_macros (gcc -O2).

### angr.analyses.decompiler.known_patterns.layouts

Runtime-specific struct layout offsets for the STL container patterns.

Where a field offset is simply the n-th pointer word it is `ctx.word(n)` (scales with the architecture). Where a runtime differs in a less regular way, the conditional lives here: the MSVC std::string keeps its size behind a fixed 16-byte SSO buffer union while libstdc++ keeps it one word past the data pointer.

angr.analyses.decompiler.known_patterns.layouts.**string_data_offset**(*_ctx*)

Offset of the character data pointer (_M_p / _Bx). 0 in both runtimes.

**Return type**
int

**Parameters**
**_ctx** (PatternContext)

angr.analyses.decompiler.known_patterns.layouts.**string_size_offset**(*ctx*)

Offset of the length field (_M_string_length / _Mysize).

**Return type**
int

**Parameters**
**ctx** (PatternContext)

angr.analyses.decompiler.known_patterns.layouts.**string_capacity_offset**(*ctx*)

libstdc++: offset of the `{ _M_local_buf[16] | _M_allocated_capacity }` union, immediately after _M_p and _M_string_length. Only libstdc++ needs it: the MSVC STL keeps _Myres in a plain field and its `capacity()` is a bare load, so no pattern here asks for the MSVC offset.

**Return type**
int

**Parameters**
**ctx** (PatternContext)

angr.analyses.decompiler.known_patterns.layouts.**vector_begin_offset**(*ctx*)

**Return type**
int

**Parameters**
**ctx** (PatternContext)

angr.analyses.decompiler.known_patterns.layouts.**vector_end_offset**(*ctx*)

**Return type**
int

**Parameters**
**ctx** (PatternContext)

angr.analyses.decompiler.known_patterns.layouts.**vector_cap_offset**(*ctx*)

**Return type**
int

**Parameters**
**ctx** (PatternContext)

angr.analyses.decompiler.known_patterns.layouts.**list_next_offset**(*ctx*)

**Return type**
int

**Parameters**
**ctx** (PatternContext)

angr.analyses.decompiler.known_patterns.layouts.**list_prev_offset**(*ctx*)

**Return type**
int

**Parameters**
**ctx** (PatternContext)

### angr.analyses.decompiler.known_patterns.libm_bits

libm floating-point bit-twiddling idioms (fabs / -x / copysign / isnan / isinf).

Every one of these "functions" is a compiler intrinsic: on x86-64 with SSE they become a single `andpd` / `xorpd` / `andnpd` / `ucomisd` against a sign-mask constant kept in `.rodata`. angr's `constant_derefs` peephole folds the rip-relative mask load into a `Const`, so what reaches the AIL is a plain integer bit operation on the raw float bits: `fabs(x)` is `x & 0x7fffffffffffffff`, unreadable unless it is named again.

Two AIL spellings show up for the SSE operand, depending on whether the 128-bit vector op survived narrowing:

- narrowed: `BinaryOp(And, vvar_0, 0x7fffffffffffffff<64>)`
- unnarrowed: `BinaryOp(Xor, Conv(64->128, vvar_0), 0x8000000000000000<128>)` under an `Extract(..., 64bits@0)`

`_sse_operand` accepts both and always captures the underlying vvar, which the Outliner requires (call parameters must bind VirtualVariables).

Intel-SSE specific (the mask widths are fixed by the FP format, not by the pointer size), so these are not word-scaled. Calibrated against tests/x86_64/decompiler/known_patterns_libm_bits (gcc -O2 -fno-tree-vectorize).

### angr.analyses.decompiler.known_patterns.linked_list

Doubly-linked-list idioms shared by Windows `LIST_ENTRY` (Flink/Blink) and Linux kernel `list_head` (next/prev): the same two-pointer layout, so one set of templates matches both WDK drivers and kernel/Boost-intrusive code.

Covers the empty test, head init, unlink (`RemoveEntryList` / `__list_del_entry`), head/tail insertion (`InsertHeadList` / `list_add`, `InsertTailList` / `list_add_tail`), the two unlink supersets the kernel actually calls, `list_del` (unlink + LIST_POISON1/2 stores) and `list_del_init` (unlink + re-init), and the `hlist_del` diamond. The supersets rely on the finder preferring the largest match, so they win over the bare unlink they contain.

The link offsets are word-scaled from the `PatternContext`, so the same definitions work on 32- and 64-bit. Opt-in (genericish C idioms). Calibrated against tests/x86_64/windows/known_patterns_wdk_list.exe and tests/x86_64/decompiler/known_patterns_linux_list (both -O2, SLP off); the compiler orders the link loads/stores freely, so the multi-statement idioms use unordered statement-set matching (`hlist_del` excepted: PGraphPat block bodies only support ordered matching).

hlist_add_head is absent on purpose: its last two stores land in the diamond's join block, and _outline_graph requires every interior block to be fully consumed, so the idiom cannot be outlined without misrepresenting it.

### angr.analyses.decompiler.known_patterns.pattern

KnownPattern: declarative metadata around a pattern-AST.

A *KnownPattern* couples a structural pattern (see *dsl*) with the information needed to turn a match into a synthesized call: the call name, the ordered parameters (mapping captures to call arguments and types), the return type, and applicability guards.

**class** angr.analyses.decompiler.known_patterns.pattern.**CppRef**

Bases: object

A reference to a C++ class type registered in the cpp::std SimTypeCollection, keyed by its unique (mangled-style) name.

**unique_name:** *str*

**ptr:** *bool* = **True**

**resolve**(*arch*)

**Return type**
*SimType* | None

**Parameters**
**arch** (*Arch*)

**__init__**(*unique_name*, *ptr=True*)

**Parameters**

- **unique_name** (*str*)
- **ptr** (*bool*)

**Return type**
None

angr.analyses.decompiler.known_patterns.pattern.**resolve_typeref**(*ref*, *arch*)

Resolve a TypeRef into a SimType: a C declaration string is parsed with parse_type; a CppRef is looked up in the cpp::std collection.

**Return type**
*SimType* | None

**Parameters**

- **ref** (*str | CppRef | None*)
- **arch** (*Arch*)

**class** angr.analyses.decompiler.known_patterns.pattern.**PatternParam**

Bases: object

One parameter of the synthesized call.

**Variables**

- **capture** – Name of the pattern capture bound to this parameter.
- **type** – Argument type for the synthesized call's prototype; flows into Typehoon as a per-argument subtype constraint.

- **type_hint** – Optional cpp::std unique name emitted as a direct vvar type hint (for by-value captures where the vvar *is* the object rather than a pointer to it).

**capture:** str

**type:** str | *CppRef* | None = None

**type_hint:** str | None = None

**__init__**(*capture*, *type=None*, *type_hint=None*)

**Parameters**

- **capture** (*str*)
- **type** (*str | CppRef | None*)
- **type_hint** (*str | None*)

**Return type**
None

class angr.analyses.decompiler.known_patterns.pattern.**KnownPattern**

Bases: object

A known code idiom, described declaratively.

**Variables**

- **name** – Identifier, e.g. "std_string_length".
- **display_name** – Human-readable name, e.g. "std::string::length".
- **call_name** – Target name of the synthesized Call.
- **pattern** – The structural pattern to match.
- **params** – Ordered parameters; their captures become the call arguments, in order.
- **returnty** – Return type of the synthesized call.
- **returnty_factory** – Optional callable building a call-site-specific return type from the values of the constant extra_args captures (e.g. CONTAINING_RECORD returns a pointer to a record whose layout depends on the matched field offset). Falls back to returnty when it returns None or when the constant values are unavailable.
- **extra_args** – Names of (typically constant) captures appended to the call arguments after params.
- **arches** – Allowed archinfo.Arch.name values; None = any.
- **platforms** – Allowed OS names (project.simos.name, lowercased); None = any.
- **where** – Optional cross-capture predicate evaluated on the bindings after a structural match.
- **collapse_capture** – Optional capture name identifying the idiom that a match belongs to: within one function, matches binding it to the same virtual variable all describe *one* source-level idiom, and only the one with the largest collapse_max_capture constant is kept. The compiler folds CONTAINING_RECORD's subtraction into every field displacement, so one source idiom otherwise emits a call per field, each naming a different (and mostly wrong) record base.
- **collapse_max_capture** – Capture whose constant value orders the collapsing; required when collapse_capture is set.

- **binary_guard** – Optional predicate on the Project; the pattern is only used when it returns True (e.g. require C++ evidence for STL patterns to avoid false positives on plain C binaries).
- **default_enabled** – Whether the pattern is used when the caller does not explicitly select patterns. Generic patterns prone to false positives should set this False.

**name:** `str`

**display_name:** `str`

**call_name:** `str`

**pattern:** *PatternExpr* | *PatternStmt* | *PGraphPat*

**params:** `tuple[`*PatternParam*`, ...]`

**returnty:** `str` | *CppRef* | `None` = `None`

**returnty_factory:** `Callable[[Arch, dict[str, int]], SimType | None] | None` = `None`

**extra_args:** `tuple[str, ...]` = `()`

**arches:** `tuple[str, ...] | None` = `None`

**platforms:** `tuple[str, ...] | None` = `None`

**where:** `Callable[[dict[str,` *Expression*`]], bool] | None` = `None`

**collapse_capture:** `str | None` = `None`

**collapse_max_capture:** `str | None` = `None`

**binary_guard:** `Callable[[`*Project*`], bool] | None` = `None`

**default_enabled:** `bool` = `True`

**suppressed_by:** `tuple[str, str, str] | None` = `None`

Drop a match of this pattern when a pattern whose name matches the regex matched on the same object: `(name regex, my capture, their capture)`. The two captures must bind `.likes()`-equal expressions. This is how a generic accessor yields to a specific one on the same base: a bare `Load(s + 8)` is std::string::length on a string and _M_finish on a vector, and the vector's other fields say which. Judged against every match of the suppressing family, selected or not, so it cannot interact with largest-match selection.

**claims_only:** `bool` = `False`

Match only to claim a base for `suppressed_by` purposes; never outlined, never rewritten, never reported. For shapes that identify an object beyond doubt but are not worth naming: a vector's capacity check says "this is a vector" and nothing a reader wants a call for.

**pure_calls:** `frozenset[str]` = `frozenset({})`

Callees this pattern reads through *PCallResult* whose only effect is their result: once the pattern has consumed that result and nothing else does, the call goes too. Naming a callee in a pattern is already a claim about what it computes; this is the claim that it computes nothing else. `__ctype_b_loc` returns a table pointer, and a bare `__ctype_b_loc();` left behind after `isspace(c)` is noise.

**applicable**(*arch_name*, *platform*)

**Return type**
`bool`

**Parameters**

- **arch_name** (*str*)
- **platform** (*str | None*)

**const_args_of_call**(*call*)

Recover the `extra_args` constant values from a synthesized call's trailing arguments. Returns None when the call does not carry them.

**Return type**
dict[str, int] | None

**Parameters**
**call** (Call)

**prototype**(*arch*, *const_args=None*)

Build the synthesized call's prototype. `const_args` (the values of the constant `extra_args` captures, see *const_args_of_call()*) enables call-site-specific return types via `returnty_factory`. Returns None if any declared type fails to resolve.

**Return type**
*SimTypeFunction* | None

**Parameters**

- **arch** (*Arch*)
- **const_args** (*dict[str, int] | None*)

**__init__**(*name*, *display_name*, *call_name*, *pattern*, *params*, *returnty=None*, *returnty_factory=None*, *extra_args=()*, *arches=None*, *platforms=None*, *where=None*, *collapse_capture=None*, *collapse_max_capture=None*, *binary_guard=None*, *default_enabled=True*, *suppressed_by=None*, *claims_only=False*, *pure_calls=frozenset({})*)

**Parameters**

- **name** (*str*)
- **display_name** (*str*)
- **call_name** (*str*)
- **pattern** (PatternExpr | PatternStmt | PGraphPat)
- **params** (*tuple[PatternParam, ...]*)
- **returnty** (*str | CppRef | None*)
- **returnty_factory** (*Callable[[Arch, dict[str, int]], SimType | None] | None*)
- **extra_args** (*tuple[str, ...]*)
- **arches** (*tuple[str, ...] | None*)
- **platforms** (*tuple[str, ...] | None*)
- **where** (*Callable[[dict[str, Expression]], bool] | None*)
- **collapse_capture** (*str | None*)
- **collapse_max_capture** (*str | None*)
- **binary_guard** (*Callable[[Project], bool] | None*)
- **default_enabled** (*bool*)
- **suppressed_by** (*tuple[str, str, str] | None*)

- **claims_only** (*bool*)
- **pure_calls** (*frozenset[str]*)

**Return type**
None

### angr.analyses.decompiler.known_patterns.posix_macros

Single-expression glibc/POSIX macros that the preprocessor inlines into every caller, so they survive into decompiled output as bare bit arithmetic.

<sys/stat.h> file-type predicates all expand through one helper:

```
#define __S_ISTYPE(mode, mask)  (((mode) & __S_IFMT) == (mask))
```

with `__S_IFMT == 0170000` and one of the seven `__S_IFxxx` values, so each predicate is a two-node expression `(mode & 0xf000) == <S_IFxxx>`. The mask is the same in every libc and in the Windows CRT (`_S_IFMT`), and the pairing of that mask with exactly one of the seven legal type values is self-guarding, so these are enabled by default.

<sys/wait.h> status decoders (glibc `bits/waitstatus.h`):

```
#define __WTERMSIG(status)     ((status) & 0x7f)
#define __WIFEXITED(status)    (__WTERMSIG(status) == 0)
#define __WIFSIGNALED(status)  (((signed char) (((status) & 0x7f) + 1) >> 1) > 0)
#define __WIFSTOPPED(status)   (((status) & 0xff) == 0x7f)
#define __WEXITSTATUS(status)  (((status) & 0xff00) >> 8)
```

Only the first three leave enough residue to identify. gcc rewrites `WIFSIGNALED` into `(signed char)((status & 0x7f) + 1) > 1` (the `>> 1` and the `> 0` fold together), which is what is matched here; the signed compare is required, since the `(signed char)` cast is the whole point of the idiom. `WIFSTOPPED` narrows to a bare `(char)status == 0x7f` (the `& 0xff` is absorbed by the byte-sized compare) and `WTERMSIG` is a bare `status & 0x7f`; both are common enough shapes to be opt-in. `WEXITSTATUS` / `WSTOPSIG` (identical macros in glibc) are not matchable at all on x86: gcc compiles `(status & 0xff00) >> 8` to a single `movzbl %ah` and the decompiler renders it as a plain byte-sized virtual variable, leaving no shift or mask to match.

`WIFSIGNALED` does not match on x86-32, and should not be made to. There the `(signed char)` cast lifts to an `Extract` rather than a `Convert`, and gcc takes the *inverted* branch, so the AIL reads

> Extract(((status & 0x7f) + 1), 8bits@0) CmpLE 1<8>

The missing `Extract` node in the DSL is the smaller half of the problem: that expression is `!WIFSIGNALED(status)`, and a KnownPattern replaces the matched expression by the value of its synthesized call, so there is no way to spell the negation. A pattern matching this shape and emitting `WIFSIGNALED(status)` would silently swap the two arms of every wait-status branch on 32-bit: the "child was killed by signal N" arm would become the "exited normally" one. The x86-32 sites (0 of 13 busybox functions) are therefore a deliberate gap until the pattern model can express a negated result.

<sys/sysmacros.h> `major()` is an `extern __inline` function that -O2 inlines; its glibc encoding scatters the major number over bits 8-19 and 32-63:

```
__major  = ((__dev & 0x00000000000fff00u) >>  8);
__major |= ((__dev & 0xfffff00000000000u) >> 32);
```

which gcc reassociates into `((dev >> 8) & 0xfff) | ((dev >> 32) & 0xfffff000)` on 64-bit targets, and into a funnel-shift pair on 32-bit ones (see `_build_major32`). That constant quartet is unique enough to enable by default. `minor()` and `makedev()` compile to `Insert`/`Extract` AIL expressions for which the pattern DSL has no node, so they are not defined here.

Every macro argument is matched as a virtual variable (optionally behind a compiler-inserted narrowing Convert), never as a Load. The consequence is that a caller reading the mode straight out of a `struct stat` field at the point of use does not match at all: `PVVar.match` returns None on a Load, so the finder never produces a match. (It does *not* reach `_rewrite_callsite` and fail there; that error can only occur for patterns whose parameter capture is not a bare PVVar.) Measured on real -O2 code this costs less than it sounds: struct stat is almost always a stack local, so `st.st_mode` is lifted to a stack vvar that matches; only pointer dereferences such as `inode->i_mode` stay Loads.

Calibrated against tests/x86_64/decompiler/known_patterns_glibc_macros (gcc 12.2 -O2; re-checked at -O1/-O3, and against glibc 2.36 headers).

`angr.analyses.decompiler.known_patterns.posix_macros.`**`macro_operand`**(*cap*)

A macro's scalar argument: a virtual variable, possibly narrowed by the compiler, or a field read.

PVVar does not see through Convert wrappers on its own, and binding the pre-narrowing vvar keeps the synthesized call argument correctly typed.

The `PLoad` arm is what makes `S_ISDIR(inode->i_mode)` matchable at all. Before the outliner could lift a parameter expression into a temporary, a capture had to be a virtual variable, so the same macro was recognized when its argument was a stack local and invisible when it was a field of a struct reached through a pointer, a split with no meaning to a reader. The mask and the type constant still carry the identification; the argument's shape never did.

**Return type**
*PChoice*

**Parameters**
**cap** (*str*)

### angr.analyses.decompiler.known_patterns.protobuf_hasbits

protobuf has-bits accessors (protoc output for optional fields):

has_field() -> _has_bits_[w] & mask (Load(m) And mask) set_has() -> _has_bits_[w] |= mask (Store(m, Load(m) Or mask)) clear_has() -> _has_bits_[w] &= ~mask (Store(m, Load(m) And ~mask))

The `_has_bits_` word is a 32-bit int regardless of architecture, so the load width is fixed at 4. Generic bitfield ops, so opt-in; the mask is captured as an extra call argument. Calibrated against tests/x86_64/decompiler/known_patterns_protobuf (g++ -O2).

### angr.analyses.decompiler.known_patterns.registry

The template registry: every `KnownPatternTemplate` the package defines, keyed by call name and by human name, plus the helpers that turn a user's selection or a target's gates into the list of templates to match with.

Kept apart from the package `__init__` so that the finder and the call-info helpers can import it at module load without a cycle.

`angr.analyses.decompiler.known_patterns.registry.`**`ALL_PATTERNS = 'all'`**

Value of the `known_patterns` decompilation option that force-enables every registered template, opt-in ones included.

**`exception`** `angr.analyses.decompiler.known_patterns.registry.`**`UnknownPatternError`**

Bases: `ValueError`

Raised when a pattern selection names a template that is not registered.

`angr.analyses.decompiler.known_patterns.registry.`**`register_pattern_template`**(*template*)

Register a KnownPatternTemplate. Call names are unique across templates (the architecture / runtime conditionals live inside each template's `build`).

**Return type**
None

**Parameters**
**template** (KnownPatternTemplate)

angr.analyses.decompiler.known_patterns.registry.**patterns_for**(*ctx*, *templates=None*, ***, *enabled_only=False*)

Instantiate the applicable templates for `ctx` into concrete KnownPatterns. `templates` defaults to the whole registry; with `enabled_only` only default-enabled templates are used.

**Return type**
list[*KnownPattern*]

**Parameters**

- **ctx** (PatternContext)
- **templates** (*Iterable[KnownPatternTemplate] | None*)
- **enabled_only** (*bool*)

angr.analyses.decompiler.known_patterns.registry.**resolve_pattern_selection**(*selection*)

Resolve a user's force-enable selection into templates.

`selection` is one of:

- `None` / empty: nothing is force-enabled (returns an empty list);
- `"all"`: every registered template, opt-in ones included;
- an iterable of template ``name``s and/or ``call_name``s (a comma-separated string is accepted too, for the string-valued decompilation option).

An unrecognized name raises *UnknownPatternError* rather than silently selecting nothing.

**Return type**
list[*KnownPatternTemplate*]

**Parameters**
**selection** (*str | Iterable[str] | None*)

angr.analyses.decompiler.known_patterns.registry.**partition_templates**(*gate_ctx*, *forced=()*, *templates=None*)

Split templates into `(enabled, deferred)` for one target.

`enabled` are the templates to match with right away: default-on ones, ones the caller force-enabled, and ones whose gate already opens on target evidence alone. `deferred` are the ones whose gate needs per-function evidence (see *CorroboratedBy*) and must be re-evaluated by the finder once the first matching stage has run.

**Return type**
tuple[list[*KnownPatternTemplate*], list[*KnownPatternTemplate*]]

**Parameters**

- **gate_ctx** (GateContext)
- **forced** (*Iterable[KnownPatternTemplate]*)
- **templates** (*Iterable[KnownPatternTemplate] | None*)

### angr.analyses.decompiler.known_patterns.std_string_cstr

Inlined MSVC `std::string::c_str()` / `data()`: the small-string optimization (SSO) select.

The MSVC STL stores the character buffer in a 16-byte union `_Bx` at offset 0 (either an inline `_Buf[16]` or, for long strings, a heap `_Ptr`), the size at offset 16, and the capacity `_Myres` one word further on. `c_str()` returns the inline buffer address when the string is short and the heap pointer when it is long, i.e.:

```
result = &s->_Bx;                 // == s   (the buffer address)
if (s->_Myres >= 16)              // capacity past the SSO threshold?
    result = s->_Bx._Ptr;        // == *(void**)s  (the heap pointer)
```

which the compiler lowers to a control-flow diamond. In clean SSA it is:

```
entry:  if (Load(s + cap_off) < 16) goto merge else goto heap
heap:   h = Load(s)
merge:  result = Phi(h [heap], s [entry])          // Phi(*s, s)
```

so it is a `PGraphPat` (entry + heap; the phi merge is the frontier) rather than an expression idiom. This matches the SSA form before de-phi, the form the KnownPatternOutliner pass sees when it runs at BEFORE_VARIABLE_RECOVERY, so the idiom is recognized and outlined during decompilation. (SSA destruction later inserts `res = s` / `h2 = h` copies that would obscure this shape; matching the clean form avoids depending on them and needs no cross-copy unification.) The offsets are word-scaled (cap_off is 24 on x64, 20 on x86); the 16-byte SSO threshold is fixed for `char` strings.

libstdc++ `c_str()` is a bare `_M_p` load (no SSO branch) and is not matched here. Opt-in, because a control-flow select is fairly generic. Calibrated against tests/x86_64/windows/known_patterns_msvc_string_cstr.exe (MSVC 19 /O2).

### angr.analyses.decompiler.known_patterns.std_string_dtor

The inlined `std::string` destructor.

`~basic_string()` is one line of source and, at -O2, five lines of machine code that the compiler emits at every scope exit of every local string and in every destructor of every class holding one. It is by a wide margin the most common inlined idiom in C++ binaries: the DWARF inline records of the -O2 benchmark corpus put `~basic_string` / `_M_dispose` / `_M_destroy` at 5.9M sites across 245 binaries, an order of magnitude above any accessor.

libstdc++ frees the buffer only when the string is not using its small-string buffer, which is exactly the SSO test `capacity()` also uses:

```
if (_M_p != &_M_local_buf)
    operator delete(_M_p, _M_allocated_capacity + 1);
```

a triangle whose taken arm is a single call. Four things have to agree for a match: the same object address in the compare and in both loads, the local buffer's offset appearing both as an address and as a field, the `+ 1`, and the callee being `operator delete`. That is what makes this pattern self-guarding enough to be on by default. The `+ 1` is the NUL terminator: libstdc++ allocated `capacity + 1` bytes and hands the same number back to the sized `operator delete`.

Sized deallocation is a C++14 feature and a compiler switch, so the one-argument `operator delete(void *)` spelling gets its own template; the two cannot share one pattern object because a statement pattern has no PChoice.

Calibrated against tests/x86_64/decompiler/known_patterns_stl5 (g++ 12.2.0 -O2).

**angr.analyses.decompiler.known_patterns.std_string_dtor.OPERATOR_DELETE = frozenset({'??3@YAXPAX@Z', '??3@YAXPAXI@Z', '??3@YAXPEAX@Z', '??3@YAXPEAX_K@Z', '_ZdlPv', '_ZdlPvSt11align_val_t', '_ZdlPvj', '_ZdlPvjSt11align_val_t', '_ZdlPvm', '_ZdlPvmSt11align_val_t', 'operator delete(void*)', 'operator delete(void*, unsigned int)', 'operator delete(void*, unsigned long)'})**

Every spelling of `operator delete` a compiler emits for this idiom. The mangled names are exact; the demangled ones are accepted too because *PCall* matches against both and a stripped-but-imported callee may only be known by one of them.

### angr.analyses.decompiler.known_patterns.std_string_internals

libstdc++ std::string internals that are not accessors: the SSO test and the two length-setting idioms.

`_M_set_length(n)` is what every mutating operation ends with. It stores the new length and re-terminates the buffer:

```
s->_M_string_length = n;
s->_M_p[n] = '\0';
```

768K DWARF inline sites across 242 binaries of the -O2 benchmark corpus, third after the destructor and the SSO test. It is self-guarding because `n` appears twice: once as the stored length and once as the offset added to the data pointer, against a base the two field displacements already pin down.

`clear()` is the same idiom with `n == 0`, and is a different pattern: with no offset to add there is no second occurrence of `n` to unify on, so the shape degrades to "zero a word at +8 and a byte through the pointer at +0". That is a plausible reset of any pointer-and-count pair, so it is opt-in and corroborated.

`_M_is_local()` is the SSO test on its own, `_M_p == &_M_local_buf`, the same comparison the destructor guards on and `capacity()` selects on. It is opt-in for a different reason: wherever it appears as part of one of those, the larger match wins and this one is redundant; what is left over is the handful of places where the test stands alone.

Calibrated against tests/x86_64/decompiler/known_patterns_stl5 (g++ 12.2.0 -O2).

### angr.analyses.decompiler.known_patterns.std_string_length

Inlined std::string::length() / size() and empty().

The length field is an 8- or 4-byte load of the string's size member. libstdc++ keeps `_M_string_length` one word past the data pointer; the MSVC STL keeps `_Mysize` behind a fixed 16-byte SSO buffer union (see *layouts*). One template covers all of {libstdc++, MSVC} x {32, 64}-bit.

Calibrated against tests/x86_64/decompiler/known_patterns_stl (g++ 12.2.0 -O2), tests/x86_64/windows/known_patterns_stl_msvc_17_x64.exe and tests/i386/windows/known_patterns_stl_msvc_17_x86.exe (VS 2022 /O2).

### angr.analyses.decompiler.known_patterns.std_swap

Inlined `std::swap`, the three-move exchange of two memory locations.

Three statement orders occur, and all three are common:

- as written, with the second value read inline:

```
t = *a;  *a = *b;  *b = t;
```

- as scheduled, with both loads hoisted, which is what any instruction scheduler produces, and what gcc emits for a struct assignment that swaps field by field:

```
t1 = *a;  t2 = *b;  *a = t2;  *b = t1;
```

- the same, storing in the other order:

```
t1 = *a;  t2 = *b;  *b = t1;  *a = t2;
```

which differs from the previous one with `a` and `b` renamed: that renaming also swaps the two loads, so the sequence differs either way round. gcc emits it for consecutive field swaps, where it lets the store to one field sit next to the load of the next.

They cannot share a pattern object, since a statement sequence is one object, so each is its own template. The width is a parameter; see `_SWAP_WIDTHS`.

The addresses are *PField* captures rather than bare variables because swapping fields is the usual case (`std::swap(lhs.n, rhs.n)` inside a move assignment): the address arrives as `obj + 8`, and the outliner materializes the pointer for the call.

Calibrated against tests/x86_64/decompiler/known_patterns_multiblock (g++ -O2).

angr.analyses.decompiler.known_patterns.std_swap.**STD_SWAP = KnownPatternTemplate(call_name='std::swap (1b)', build=<function _make_swap_template.<locals>.build>, arches=frozenset({'AMD64', 'X86'}), languages=frozenset({'cpp'}), runtimes=None, platforms=None, default_enabled=False, name='std_swap_1', gate=None)**

the word-sized, as-written spelling, the one the fixture tests

### angr.analyses.decompiler.known_patterns.std_vector_size

Inlined std::vector<T>::size() and capacity().

libstdc++ and the MSVC STL both lay out std::vector as three pointers (_M_start/_M_finish/_M_end_of_storage), so both accessors are one pointer difference divided by sizeof(T):

size() == (_M_finish - _M_start) / sizeof(T) capacity() == (_M_end_of_storage - _M_start) / sizeof(T)

i.e. the same shape reading a different minuend field. The field offsets and load width come from the `PatternContext` (word-scaled), so one definition covers 32- and 64-bit.

`sizeof(T)` is not recoverable from anything but the divisor, so there is one template per element size. Three ranges of sizeof(T) need three spellings:

- a power of two up to 8: a bare shift, and there is a named C++ type of that width (short/int/long long);
- a power of two above 8: still a bare shift, but no named type, so the element is the opaque `T<N>` class (`sizeof(std::string) == 32`, and `std::vector<std::string>` is the corpus' most common element type);
- not a power of two: the exact-division magic-multiply idiom below.

`sizeof(T) == 1` is not covered for size()/capacity(): with no shift the idiom is a bare pointer difference, too generic to name.

Calibrated against the known_patterns_stl binaries (libstdc++ ELF, MSVC x64/x86).

angr.analyses.decompiler.known_patterns.std_vector_size.**exact_div_magic**(*elt_size*, *bits*)

`(shift, magic)` of the exact-division-by-`elt_size` idiom.

**Return type**
tuple[int, int]

**Parameters**

- **elt_size** (*int*)
- **bits** (*int*)

angr.analyses.decompiler.known_patterns.std_vector_size.**make_std_vector_size_template**(*elt_name*, *log2_elt_size*)

A power-of-two `std::vector<elt_name>::size` template.

**Parameters**

- **elt_name** (*str*)
- **log2_elt_size** (*int*)

angr.analyses.decompiler.known_patterns.std_vector_size.**make_std_vector_size_exactdiv_template**(*elt_name*, *elt_size*)

A non-power-of-two `std::vector<elt_name>::size` template.

**Parameters**

- **elt_name** (*str*)
- **elt_size** (*int*)

angr.analyses.decompiler.known_patterns.std_vector_size.**STD_VECTOR_EXACTDIV_SIZES** = (3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511)

Every opaque element size, power-of-two and not. The list is wide on purpose: sizeof(T) is whatever the program's record happens to be (112 for CryptoPP::ECPPoint, say), and a size that is not covered leaves the raw magic multiply in the output, which is the shape this pattern exists to remove.

angr.analyses.decompiler.known_patterns.std_vector_size.**STD_VECTOR_STACK_TEMPLATES = [KnownPatternTemplate(call_name='std::vector<short>::size (stack)', build=<function _make_vector_div_template.<locals>.build>, arches=frozenset({'AMD64', 'X86'}), languages=frozenset({'cpp'}), runtimes=None, platforms=None, default_enabled=True, name='std_vector_short_size_stack', gate=None), KnownPatternTemplate(call_name='std::vector<int>::size (stack)', build=<function _make_vector_div_template.<locals>.build>, arches=frozenset({'AMD64', 'X86'}), languages=frozenset({'cpp'}), runtimes=None, platforms=None, default_enabled=True, name='std_vector_int_size_stack', gate=None), KnownPatternTemplate(call_name='std::vector<long long>::size (stack)', build=<function _make_vector_div_template.<locals>.build>, arches=frozenset({'AMD64', 'X86'}), languages=frozenset({'cpp'}), runtimes=None, platforms=None, default_enabled=True, name='std_vector_long_long_size_stack', gate=None)]**

same shape, slots instead of loads. Only the named element sizes; the opaque T<N> set would double a four-figure library for a shape that is only reachable where the object never escapes to a pointer.

**Type**
Stack twins of *size*()

### angr.analyses.decompiler.known_patterns.stl_accessors2

C++ STL accessors, round two: std::string capacity/back/front and std::vector<T>::back().

`std::string::capacity()` is the only control-flow idiom here. libstdc++ lays std::string out as `{ _M_p , _M_string_length , union { _M_local_buf[16] | _M_allocated_capacity } }`, so `capacity()` is the SSO select:

```
_M_p == &_M_local_buf ? 15 : _M_allocated_capacity
```

i.e. a diamond. Unlike the MSVC `c_str()` triangle (see *std_string_cstr*), both arms of this diamond assign a value, so ITERegionConverter, which runs at AFTER_GLOBAL_SIMPLIFICATION, before the KnownPatternOutliner at BEFORE_VARIABLE_RECOVERY, has already folded the region into a single `x = c ? a : b` assignment by the time patterns are matched. The idiom is therefore a *PITE* expression pattern, not a PGraphPat. (When gcc's tail duplication copies the join into both arms instead of merging them, no ITE and no single-exit region exists and the idiom is not matched; that is intended, since there is nothing to outline.)

The comparison `Load(s) == s + 16`, a loaded pointer tested against the object's own address plus the union offset, is what makes the pattern self-guarding; the SSO threshold 15 is fixed for `char` strings, and the union offset is word-scaled, so one definition covers 32- and 64-bit.

`back()`/`front()` are plain expression idioms: `*(_M_p + size - 1)` and `*_M_p`. `std::vector<T>::back()` is `*(_M_finish - sizeof(T))`.

Calibrated against tests/x86_64/decompiler/known_patterns_stl2 (g++ 12.2.0 -O2).

angr.analyses.decompiler.known_patterns.stl_accessors2.**make_std_vector_back_template**(*elt_name*, *elt_size*, *returnty*)

A `std::vector<elt_name>::back` template: `*(_M_finish - sizeof(T))`.

**Parameters**

- **elt_name** (*str*)
- **elt_size** (*int*)
- **returnty** (*str*)

### angr.analyses.decompiler.known_patterns.stl_containers

C++ STL std::vector accessors beyond size()/capacity(): empty / operator[].

The three-pointer vector layout is shared by libstdc++ and the MSVC STL, so a single template per accessor, with word-scaled offsets and pointer-sized loads from the `PatternContext`, covers 32- and 64-bit on either runtime.

`capacity()` lives in *std_vector_size* with `size()`: the two are one expression against different fields, and both need a template per element size.

Calibrated against the known_patterns_stl binaries (libstdc++ ELF, MSVC x64/x86).

`angr.analyses.decompiler.known_patterns.stl_containers.`**`make_std_vector_index_template`**(*elt_name*, *elt_size*, *returnty*)

A `std::vector<elt_name>::operator[]` template: `*(_M_start + i * sizeof(T))`.

One per element size. Only the 4-byte (int) template existed, which covers 5% of the corpus' DWARF sites for this accessor: 78% of them are 8-byte elements (`unsigned long long` alone is 30%, pointers another 14%).

The scale is matched as `Mul` *or* `Shl`: a power-of-two element size lifts from *lea (%rax,%rdx,8)* either way depending on how the index reached the address, and pinning one spelling silently halves the coverage.

**Parameters**

- **elt_name** (*str*)
- **elt_size** (*int*)
- **returnty** (*str*)

### angr.analyses.decompiler.known_patterns.templates

KnownPatternTemplate: a single, context-parameterized definition of an idiom that instantiates itself into a concrete `KnownPattern` for a target's `PatternContext`.

One template replaces the per-architecture / per-runtime copies of a pattern: its `build(ctx)` computes the concrete field offsets, load sizes, and return types from the context, so the same definition covers 32- and 64-bit, libstdc++ and MSVC, and so on.

**class** `angr.analyses.decompiler.known_patterns.templates.`**KnownPatternTemplate**

Bases: `object`

A context-aware pattern definition.

**Variables**

- **call_name** – The synthesized call name (unique across templates).
- **build** – `build(ctx) -> KnownPattern | None`; emits the concrete pattern for `ctx` (None if it cannot).
- **arches** – Allowed `Arch.name` values; None = any.
- **languages** – Allowed languages ({"c","cpp"}); None = any.
- **runtimes** – Allowed C++ runtimes ({"libstdcxx","msvc"}); None = any.
- **platforms** – Allowed platforms ({"linux","windows"}); None = any.
- **default_enabled** – Whether the template is used when the caller does not explicitly select patterns.

- **gate** – Optional *PatternGate* that turns an opt-in template on for targets (or functions) where the evidence says the idiom is real; see *gating*. A gate can only open a template, never close a default-on one.

**call_name:** str

**build:** Callable[[*PatternContext*], *KnownPattern* | None]

**arches:** frozenset[str] | None = None

**languages:** frozenset[str] | None = None

**runtimes:** frozenset[str] | None = None

**platforms:** frozenset[str] | None = None

**default_enabled:** bool = False

**name:** str = ''

**gate:** *PatternGate* | None = None

**enabled_for**(*gate_ctx*)

**Return type**
bool

**Parameters**
**gate_ctx** (GateContext)

**applicable**(*ctx*)

**Return type**
bool

**Parameters**
**ctx** (PatternContext)

**instantiate**(*ctx*)

The concrete pattern for `ctx`, or None if this template does not apply there. Memoized: a KnownPattern is a frozen tree that depends on nothing but the context, and callers treat it read-only.

**Return type**
*KnownPattern* | None

**Parameters**
**ctx** (PatternContext)

**__init__**(*call_name*, *build*, *arches=None*, *languages=None*, *runtimes=None*, *platforms=None*, *default_enabled=False*, *name=''*, *gate=None*, *_cache=<factory>*)

**Parameters**

- **call_name** (*str*)
- **build** (*Callable[[PatternContext], KnownPattern | None]*)
- **arches** (*frozenset[str] | None*)
- **languages** (*frozenset[str] | None*)
- **runtimes** (*frozenset[str] | None*)
- **platforms** (*frozenset[str] | None*)

- **default_enabled** (*bool*)
- **name** (*str*)
- **gate** (PatternGate | *None*)
- **_cache** (*dict*)

**Return type**
None

angr.analyses.decompiler.known_patterns.templates.**make_template**(*call_name*, *build*, *, *arches=None*, *languages=None*, *runtimes=None*, *platforms=None*, *default_enabled=False*, *name=''*, *gate=None*)

**Return type**
*KnownPatternTemplate*

**Parameters**

- **call_name** (*str*)
- **build** (*Callable[[PatternContext], KnownPattern | None]*)
- **arches** (*frozenset[str] | tuple[str, ...] | None*)
- **languages** (*frozenset[str] | tuple[str, ...] | None*)
- **runtimes** (*frozenset[str] | tuple[str, ...] | None*)
- **platforms** (*frozenset[str] | tuple[str, ...] | None*)
- **default_enabled** (*bool*)
- **name** (*str*)
- **gate** (PatternGate | *None*)

### angr.analyses.decompiler.known_patterns.vector_claims

Claims-only std::vector fingerprints: shapes that identify a vector beyond doubt but are not worth a call of their own.

They exist for one reason. `std::string::length` is a bare word load at +8, and so is `std::vector::_M_finish`: 63% of `length`'s false positives on the benchmark corpus sit in functions with a vector at that very site. The generated `size()/capacity()` templates already claim a base wherever they match, but they need a shift or a magic multiply to key on; two common vector shapes carry neither, and are matched here purely so the base counts as a vector's (see *claims_only*):

- `Load(v + 8) - Load(v)`: `size()` of a one-byte element type, which no size() template covers because there is nothing to divide by;
- `Load(v + 8) ==/!= Load(v + 16)`: `push_back`'s capacity check, `_M_finish == _M_end_of_storage`, the shape behind the review's own example of a vector named as a string.

A string never subtracts its length from its data pointer, and compares its length with its capacity only in `size() == capacity()`, which is rare enough inline to accept.

### angr.analyses.decompiler.known_patterns.vector_math

3-component float vector math (glm / DirectXMath / game-graphics code).

Scalar float arithmetic compiles to SSE MulV/AddV on 128-bit registers with Conv(32->128, Load(..., 4)) operands under Extract(..., 64@0). Intel-SSE specific (float width is 4 bytes regardless of pointer size), so these are not word-scaled. Opt-in; calibrated against tests/x86_64/decompiler/known_patterns_vecmath (g++ -O2 -fno-tree-vectorize).

### angr.analyses.decompiler.known_patterns.wdk_shared_data

Reads of `KUSER_SHARED_DATA`, the page Windows maps read-only at the fixed virtual address `0x7FFE0000` in every user-mode process (x86, x64 and ARM64 alike). ntdll, kernel32 and countless drivers read its fields through the `SharedUserData` macro of ntddk.h/wdm.h, which compiles to a bare load from an absolute constant address:

```
SharedUserData->NtMajorVersion   ==   *(volatile ULONG *)0x7FFE026C
```

Kernel-mode code reaches the same page through `KI_USER_SHARED_DATA` (`0xFFDF0000` on x86, `0xFFFFF78000000000` on x64), so each field's pattern accepts the user- and the arch-appropriate kernel-mode address.

The absolute address is the guard here: there is no plausible non-Windows meaning for a load from `0x7FFE0000 + small offset`, so these are default-on and take no parameters (`params=()`); the synthesized call is a nullary accessor whose return type is the field's type.

Field offsets are those of the WDK `ddk/ntddk.h` `KUSER_SHARED_DATA` (cross-checked with `offsetof` on i686 and x86_64, and against Geoff Chappell's version-annotated layout); only offsets that have been stable from Windows XP through Windows 11 are default-on. Calibrated against tests/x86_64/windows/known_patterns_wdk_ksud.exe (-O2, mingw).

**class** angr.analyses.decompiler.known_patterns.wdk_shared_data.**KsudField**

Bases: object

One KUSER_SHARED_DATA field access: its byte offset, the load width used to read it, and the accessor's name.

`stable` is False for offsets that moved between Windows versions; those templates exist but are not enabled by default.

**offset:** int

**name:** str

**size:** int

**stable:** bool = True

**__init__**(*offset*, *name*, *size*, *stable=True*)

**Parameters**

- **offset** (*int*)
- **name** (*str*)
- **size** (*int*)
- **stable** (*bool*)

**Return type**
None

angr.analyses.decompiler.known_patterns.wdk_shared_data.**make_shared_user_data_template**(*field*)

A SharedUserData-> field-read template for one KUSER_SHARED_DATA field: a load of `field.size` bytes from the field's fixed address.

**Parameters**
**field** (KsudField)

angr.analyses.decompiler.known_patterns.wdk_shared_data.**make_ntstatus_severity_template**(*macro*, *severity*)

An NTSTATUS severity-predicate template: `(s >> 30) == severity`, in either the literal shift form or the mask form the compilers prefer.

**Parameters**

- **macro** (*str*)
- **severity** (*int*)

### angr.analyses.decompiler.label_collector

**class** angr.analyses.decompiler.label_collector.**LabelCollector**

Bases: object

Collect all labels.

**__init__**(*node*)

### angr.analyses.decompiler.node_replacer

**class** angr.analyses.decompiler.node_replacer.**NodeReplacer**

Bases: *SequenceWalker*

Replaces nodes in a node with new nodes based on a mapping.

**__init__**(*root*, *replacements*)

**Parameters**

- **root** (BaseNode)
- **replacements** (*dict*)

**Return type**
None

### angr.analyses.decompiler.notes

**class** angr.analyses.decompiler.notes.**DecompilationNote**

Bases: object

Describes a note that is generated during decompilation.

Key is a unique string for the decompilation note. It is used as an index in the decompilation notes dictionary in the Decompiler class. Name is string for display by default. Content is the actual content of the note. It can be of any time, but for custom types, you must override *__str__* so that it can be displayed. Level is the level of the note. The following values are available: DecompilationNoteLevel.DEBUG, DecompilationNoteLevel.INFO, DecompilationNoteLevel.WARNING, and DecompilationNoteLevel.CRITICAL.

__init__(*key*, *name*, *content*, *, *level=DecompilationNoteLevel.INFO*)

**Parameters**

- **key** (*str*)
- **name** (*str*)
- **content** (*Any*)

**key**

**name**

**content**

**level**

**to_jsonable**()

**Return type**
dict[str, Any]

**to_json**()

**Return type**
str

**classmethod from_jsonable**(*d*)

**Return type**
*DecompilationNote*

**Parameters**
**d** (*dict[str, Any]*)

**classmethod from_json**(*s*)

**Return type**
*DecompilationNote*

**Parameters**
**s** (*str*)

**class** angr.analyses.decompiler.notes.**DecompilationNoteLevel**

Bases: Enum

Enum class describing the level of each decompilation note.

**DEBUG = 0**

**INFO = 1**

**WARNING = 2**

**CRITICAL = 3**

**class** angr.analyses.decompiler.notes.**DeobfuscatedString**

Bases: object

Represents a deobfuscated string.

**__init__**(*value*, *obf_type*, *ref_addr=None*)

**Parameters**

- **value** (bytes)
- **obf_type** (*str*)
- **ref_addr** (*int | None*)

**value**

**type**

**ref_addr**

**class** angr.analyses.decompiler.notes.**DeobfuscatedStringsNote**

Bases: *DecompilationNote*

Represents a decompilation note that describes obfuscated strings found during decompilation.

**__init__**(*key='deobfuscated_strings'*, *name='Deobfuscated Strings'*)

**Parameters**

- **key** (*str*)
- **name** (*str*)

**strings:** dict[int, *DeobfuscatedString*]

**add_string**(*obf_type*, *value*, ***, *ref_addr*)

Add a deobfuscated string to the note.

**Parameters**

- **obf_type** (str) – The type of obfuscation (e.g., “1”, “2”).
- **value** (bytes) – The deobfuscated string value.
- **ref_addr** (int) – The address where this string is referenced, if applicable.

**to_jsonable**()

**Return type**

dict[str, Any]

**content**

**key**

**level**

**name**

## Submodules

*decompilation_note*
*deobfuscated_strings*

### angr.analyses.decompiler.notes.decompilation_note

class angr.analyses.decompiler.notes.decompilation_note.**DecompilationNoteLevel**

Bases: Enum

Enum class describing the level of each decompilation note.

**DEBUG = 0**

**INFO = 1**

**WARNING = 2**

**CRITICAL = 3**

class angr.analyses.decompiler.notes.decompilation_note.**DecompilationNote**

Bases: object

Describes a note that is generated during decompilation.

Key is a unique string for the decompilation note. It is used as an index in the decompilation notes dictionary in the Decompiler class. Name is string for display by default. Content is the actual content of the note. It can be of any time, but for custom types, you must override *__str__* so that it can be displayed. Level is the level of the note. The following values are available: DecompilationNoteLevel.DEBUG, DecompilationNoteLevel.INFO, DecompilationNoteLevel.WARNING, and DecompilationNoteLevel.CRITICAL.

**__init__**(*key*, *name*, *content*, ***, *level=DecompilationNoteLevel.INFO*)

**Parameters**

- **key** (*str*)
- **name** (*str*)
- **content** (*Any*)

**key**

**name**

**content**

**level**

**to_jsonable**()

**Return type**
dict[str, Any]

**to_json**()

**Return type**
str

classmethod **from_jsonable**(*d*)

**Return type**
*DecompilationNote*

**Parameters**
**d** (*dict[str, Any]*)

classmethod **from_json**(*s*)

**Return type**
*DecompilationNote*

**Parameters**
**s** (*str*)

### angr.analyses.decompiler.notes.deobfuscated_strings

class angr.analyses.decompiler.notes.deobfuscated_strings.**DeobfuscatedString**

Bases: object

Represents a deobfuscated string.

**__init__**(*value*, *obf_type*, *ref_addr=None*)

**Parameters**

- **value** (bytes)
- **obf_type** (*str*)
- **ref_addr** (*int | None*)

**value**

**type**

**ref_addr**

class angr.analyses.decompiler.notes.deobfuscated_strings.**DeobfuscatedStringsNote**

Bases: *DecompilationNote*

Represents a decompilation note that describes obfuscated strings found during decompilation.

**__init__**(*key='deobfuscated_strings'*, *name='Deobfuscated Strings'*)

**Parameters**

- **key** (*str*)
- **name** (*str*)

**strings:** dict[int, *DeobfuscatedString*]

**add_string**(*obf_type*, *value*, ***, *ref_addr*)

Add a deobfuscated string to the note.

**Parameters**

- **obf_type** (str) – The type of obfuscation (e.g., “1”, “2”).
- **value** (bytes) – The deobfuscated string value.
- **ref_addr** (int) – The address where this string is referenced, if applicable.

**to_jsonable**()

**Return type**
dict[str, Any]

**content**

**key**

**level**

**name**

### angr.analyses.decompiler.optimization_pass_registry

Stable name resolution for decompiler optimization passes and peephole optimizations.

angr.analyses.decompiler.optimization_pass_registry.**name_to_pass**(*name*)

Resolve a class name back to its registered pass class. Returns None for names that are not registered at the time this method is called. Peephole passes might be defined by analyses or plugins that have not yet been imported.

**Return type**
type | None

**Parameters**
**name** (*str*)

angr.analyses.decompiler.optimization_pass_registry.**pass_to_name**(*cls*)

Return a stable string identifier (the class name) for an optimization pass or peephole class.

**Return type**
str

**Parameters**
**cls** (type)

### angr.analyses.decompiler.optimization_passes

**class** angr.analyses.decompiler.optimization_passes.**BasePointerSaveSimplifier**

Bases: *OptimizationPass*

Removes the effects of base pointer stack storage at function invocation and restoring at function return.

**ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']**

**PLATFORMS = None**

**STAGE = 6**

**NAME = 'Simplify base pointer saving'**

**DESCRIPTION = 'Removes the effects of base pointer stack storage at function invocation and restoring at function return.'**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**CallStatementRewriter**

Bases: *OptimizationPass*

Rewrite call statements to assignments if needed.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 5**

**NAME = 'Unify call statements on demand.'**

**DESCRIPTION = 'Rewrite call statements to assignments if needed.'**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**CodeMotionOptimization**

Bases: *OptimizationPass*

Moves common statements out of blocks that share the same predecessors or the same successors. This is done to reduce the number of statements in a block and to make the blocks more similar to each other.

As an example:

```
if (x) {
    b = 2;
    a = 1;
    c = 3;
} else {
    b = 2;
    c = 3;
}
```

Will be turned into:

```
if (x) {
    a = 1;
}
b = 2;
c = 3;
```

Current limitations (for very conservative operations):

- moving statements above conditional jumps is not supported
- only immediate children and parents are considered for moving statements
- when moving statements down, a block is only considered if already has a matching statement at the end

**ARCHES = None**

**PLATFORMS = None**

**NAME = 'Merge common statements in sub-scopes'**

**STAGE = 6**

**DESCRIPTION = '\n Moves common statements out of blocks that share the same predecessors or the same\n successors. This is done to reduce the number of statements in a block and to make the\n blocks more similar to each other.\n\n As an example::\n\n if (x) {\n b = 2;\n a = 1;\n c = 3;\n } else {\n b = 2;\n c = 3;\n }\n\n Will be turned into::\n\n if (x) {\n a = 1;\n }\n b = 2;\n c = 3;\n\n Current limitations (for very conservative operations):\n\n - moving statements above conditional jumps is not supported\n - only immediate children and parents are considered for moving statements\n - when moving statements down, a block is only considered if already has a matching statement at the end\n '**

__init__(**args*, *max_iters=10*, *node_idx_start=0*, ***kwargs*)

**Parameters**
**node_idx_start** (*int*)

static update_graph_with_super_edits(*original_graph*, *super_graph*, *updated_blocks*)

This function updates an graph when doing block edits on a supergraph version of that same graph. The updated blocks must be provided as a dictionary where the keys are original block in the supergraph and the values are the new blocks that should replace them.

The supergraph MUST be generated using the to_ail_supergraph function, since it stores the original nodes each super node represents. This is necessary to update the original graph with the new super nodes.

**Return type**
bool

**Parameters**

- **original_graph** (*DiGraph*)
- **super_graph** (*DiGraph*)
- **updated_blocks** (*dict[Block, Block]*)

class angr.analyses.decompiler.optimization_passes.**ConditionConstantPropagation**

Bases: *OptimizationPass*

Reason about constant propagation opportunities from conditionals and propagate constants in the graph accordingly.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 2**

**NAME = 'Propagate constants using information deduced from conditionals.'**

**DESCRIPTION = 'Reason about constant propagation opportunities from conditionals and propagate constants in the graph accordingly.'**

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**ConstPropOptReverter**

Bases: *OptimizationPass*

This optimization reverts the effects of constant propagation done by the compiler as discussed in the USENIX 2024 paper SAILR. This optimization's main goal is to enable later optimizations that rely on symbolic variables to be more effective. This optimization pass will convert two statements with a difference of a const and a symbolic variable into two statements with the symbolic variables.

As an example: x = 75 puts(x) puts(75)

will be converted to: x = 75 puts(x) puts(x)

**ARCHES = None**

**PLATFORMS = None**

**STRUCTURING = ['sailr', 'dream']**

**STAGE = 10**

NAME = 'Revert Constant Propagation Optimizations'

DESCRIPTION = "This optimization reverts the effects of constant propagation done by the compiler as discussed in the\n USENIX 2024 paper SAILR. This optimization's main goal is to enable later optimizations that rely on\n symbolic variables to be more effective. This optimization pass will convert two statements with a difference of\n a const and a symbolic variable into two statements with the symbolic variables.\n\n As an example:\n x = 75\n puts(x)\n puts(75)\n\n will be converted to:\n x = 75\n puts(x)\n puts(x)"

__init__(**args*, *region_identifier=None*, *reaching_definitions=None*, ***kwargs*)

static find_conflicting_call_args(*call0*, *call1*)

**Parameters**

- **call0** (Call)
- **call1** (Call)

class angr.analyses.decompiler.optimization_passes.**ConstantDereferencesSimplifier**

Bases: *OptimizationPass*

Makes the following simplifications:

```
*(*(const_addr))  ==>  *(value) iff  *const_addr == value
```

ARCHES = None

PLATFORMS = None

STAGE = 2

NAME = 'Simplify constant dereferences'

DESCRIPTION = 'Makes the following simplifications::\n\n *(*(const_addr)) ==> *(value) iff *const_addr == value'

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**CrossJumpReverter**

Bases: *StructuringOptimizationPass*

This is an implementation to revert the compiler optimization Cross Jumping, an ISC optimization discussed in the USENIX 2024 paper SAILR. This optimization is somewhat aggressive and as such should be run last in your decompiler deoptimization chain. This deoptimization will take any goto it finds and attempt to duplicate its target block if its target only has one outgoing edge.

There are some heuristics in place to prevent duplication everywhere. First, this deoptimization will only run a max of max_opt_iters times. Second, it will not duplicate a block with too many calls.

STAGE = 10

NAME = 'Duplicate linear blocks with gotos'

DESCRIPTION = 'This is an implementation to revert the compiler optimization Cross Jumping, an ISC optimization discussed\nin the USENIX 2024 paper SAILR. This optimization is somewhat aggressive and as such should be run last in your\ndecompiler deoptimization chain. This deoptimization will take any goto it finds and attempt to duplicate its\ntarget block if its target only has one outgoing edge.\n\nThere are some heuristics in place to prevent duplication everywhere. First, this deoptimization will only run\na max of max_opt_iters times. Second, it will not duplicate a block with too many calls.'

__init__(**args*, *node_idx_start=0*, *max_opt_iters=3*, *max_call_duplications=1*, ***kwargs*)

**Parameters**

- **node_idx_start** (*int*)
- **max_opt_iters** (*int*)
- **max_call_duplications** (*int*)

class angr.analyses.decompiler.optimization_passes.**DeadblockRemover**

Bases: *OptimizationPass*

Removes condition-unreachable blocks from the graph.

ARCHES = None

PLATFORMS = None

STAGE = 9

NAME = 'Remove blocks with unsatisfiable conditions'

DESCRIPTION = 'Removes condition-unreachable blocks from the graph.'

__init__(**args*, *node_cutoff=200*, ***kwargs*)

**Parameters**
**node_cutoff** (*int*)

class angr.analyses.decompiler.optimization_passes.**DivSimplifier**

Bases: *OptimizationPass*

Simplifies various division optimizations back to “div”.

ARCHES = ['X86', 'AMD64', 'ARMCortexM', 'ARMHF', 'ARMEL']

PLATFORMS = None

STAGE = 6

NAME = 'Simplify arithmetic division'

DESCRIPTION = 'Simplifies various division optimizations back to "div".'

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**DuplicationReverter**

Bases: *StructuringOptimizationPass*

This (de)optimization reverts the effects of many compiler optimizations that cause code duplication in the decompilation. This deoptimization is the implementation of the USENIX 2024 paper SAILR’s ISD doptimization. As such, the main goal of this optimization is to remove code duplication by merging semantically similar blocks in the AIL graph.

**NAME = 'Revert Statement Duplication Optimizations'**

**DESCRIPTION = "This (de)optimization reverts the effects of many compiler optimizations that cause code duplication in\n the decompilation. This deoptimization is the implementation of the USENIX 2024 paper SAILR's ISD\n doptimization. As such, the main goal of this optimization is to remove code duplication by merging\n semantically similar blocks in the AIL graph."**

**__init__**(**args*, *max_guarding_conditions=4*, ***kwargs*)

**static boolean_operators_in_condition**(*condition*)

TODO: this entire boolean checking semantic we use needs to be removed, see how it is used for other dels needed we need to replace it with a boolean variable insertion on both branches that lead to the new block.

Say we have:

```
if (A()) {
    do_thing();
}
if (B()) {
    do_thing():
}
```

We want to translate it to:

```
int should_do_thing = 0;
if (A())
    should_do_thing = 1;
if (B())
    should_do_thing = 1;

if (should_do_thing):
    do_thing();
```

Although longer, this code can be optimized to look like:

```
int should_do_thing = A() || B();
if (should_do_thing)
    do_thing();
```

**Parameters**
**condition** (Expression)

**stmt_can_move_to**(*stmt*, *block*, *new_idx*, *io_finder=None*)

**maximize_similarity_of_blocks**(*block1*, *block2*, *graph*)

This attempts to rearrange the order of statements in block1 and block2 to maximize the similarity between them. This implementation is a little outdated since CodeMotion optimization was implemented, but it should be disabled until we have a good SSA implementation.

TODO: reimplement me when we have better SSA

**Return type**
tuple[Block, Block]

**create_merged_subgraph**(*blocks*, *graph*, *maximize_similarity=False*)

**Return type**
*AILMergeGraph*

**Parameters**
**graph** (*DiGraph*)

**similar_conditional_when_single_corrected**(*block1*, *block2*, *graph*)

**Parameters**

- **block1** (*Block*)
- **block2** (*Block*)
- **graph** (*DiGraph*)

**collect_conditions_between_nodes**(*graph*, *source*, *sinks*, *max_depth=15*)

**Parameters**

- **source** (*Block*)
- **sinks** (*list[Block]*)

**shared_common_conditional_dom**(*nodes*, *graph*)

Takes n nodes and returns True only if all the nodes are dominated by the same node, which must be a ConditionalJump

@param nodes: @param graph: @return:

**Parameters**
**graph** (*DiGraph*)

class angr.analyses.decompiler.optimization_passes.**EagerStdStringConcatenationPass**

Bases: *OptimizationPass*

Concatenate multiple constant std::string creation calls into one when possible.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 7**

**NAME = 'Condense multiple constant std::string creation calls into one when possible'**

**DESCRIPTION = 'Concatenate multiple constant std::string creation calls into one when possible.'**

**__init__**(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**ExprOpSwapper**

Bases: *SequenceOptimizationPass*

Swap operands (and the operator accordingly) in a BinOp expression.

**ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']**

**PLATFORMS = ['windows', 'linux', 'cgc']**

**STAGE = 11**

**NAME = 'Swap operands of expressions as requested'**

**DESCRIPTION = 'Swap operands (and the operator accordingly) in a BinOp expression.'**

**__init__**(**args*, *binop_operators=None*, ***kwargs*)

**Parameters**
**binop_operators** (*dict[OpDescriptor, str] | None*)

**class** angr.analyses.decompiler.optimization_passes.**FlipBooleanCmp**

Bases: *SequenceOptimizationPass*

In the scenario in which a false node has no apparent successors, flip the condition on that if-stmt. This is only useful when StructuredCodeGenerator has simplify_else_scopes enabled, as this will allow the flipped if-stmt to remove the redundant else.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 11**

**NAME = 'Flip small ret booleans'**

**DESCRIPTION = 'When false node has no successors, flip condition so else scope can be simplified later'**

**__init__**(**args*, *flip_size=9*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**ITEExprConverter**

Bases: *OptimizationPass*

Transform specific expressions into If-Then-Else expressions, or tertiary expressions in C when given a single-use expression address. Requires outside analysis to provide the target expressions.

**ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']**

**PLATFORMS = ['windows', 'linux', 'cgc']**

**STAGE = 10**

**NAME = 'Transform single-use expressions that were assigned to in different If-Else branches into ternary expressions'**

**DESCRIPTION = 'Transform specific expressions into If-Then-Else expressions, or tertiary expressions in C when\n given a single-use expression address. Requires outside analysis to provide the target expressions.'**

**__init__**(**args*, *ite_exprs=None*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**ITERegionConverter**

Bases: *OptimizationPass*

Transform regions of the form *if (c) {x = a} else {x = b}* into $x = c\ ?\ a : b$.

**ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']**

**PLATFORMS = ['windows', 'linux', 'cgc']**

**STAGE = 6**

**NAME = 'Transform ITE-assignment regions into ternary expression assignments'**

**DESCRIPTION = 'Transform regions of the form `if (c) {x = a} else {x = b}` into `x = c ? a : b`.'**

**__init__**(**args*, *max_updates=10*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**InlinedMemcpySimplifier**

Bases: *OptimizationPass*

Simplifies inlined data copying logic into calls to memcpy.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 3**

**NAME = 'Simplify inlined memcpy'**

**DESCRIPTION = 'Simplify inlined memcpy patterns into memcpy calls'**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**InlinedMemcpySimplifierLate**

Bases: *InlinedMemcpySimplifier*

Same as InlinedMemcpySimplifier but runs after SSA level 1 transformation.

**STAGE = 4**

**NAME = 'Simplify inlined memcpy (late)'**

**class** angr.analyses.decompiler.optimization_passes.**InlinedMemsetSimplifier**

Bases: *OptimizationPass*

Simplifies inlined memory setting logic into calls to memset.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 3**

**NAME = 'Simplify inlined memset'**

**DESCRIPTION = 'Simplify inlined memset patterns into memset calls'**

**MIN_ASSIGNMENTS = 2**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**InlinedMemsetSimplifierLate**

Bases: *InlinedMemsetSimplifier*

Same as InlinedMemsetSimplifier but runs after SSA level 1 transformation.

**STAGE = 4**

NAME = 'Simplify inlined memset (late)'

class angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier

Bases: OptimizationPass

Simplifies inlined string copying logic into calls to strcpy/strncpy, and consolidates multiple consecutive inlined strcpy calls.

ARCHES = None

PLATFORMS = None

STAGE = 3

NAME = 'Simplify inlined strcpy'

DESCRIPTION = 'Simplify inlined strcpy patterns and consolidate multiple inlined strcpy calls'

__init__(*args*, ***kwargs*)

static is_integer_likely_a_string(*v*, *size*, *endness*, *min_length=4*)

is_inlined_strcpy(*stmt*)

class angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifierLate

Bases: InlinedStrcpySimplifier

Same as InlinedStrcpySimplifier but runs after SSA level 1 transformation.

STAGE = 4

NAME = 'Simplify inlined strcpy (late)'

class angr.analyses.decompiler.optimization_passes.InlinedStringTransformationSimplifier

Bases: OptimizationPass

Simplifies inlined string transformation routines.

ARCHES = None

PLATFORMS = None

STAGE = 3

NAME = 'Simplify string transformations'

DESCRIPTION = 'Simplify string transformations that are commonly used in obfuscated functions.'

__init__(*args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.InlinedStrlenSimplifier

Bases: OptimizationPass

Abstracts inlined strlen functions into strlen() calls.

ARCHES = None

PLATFORMS = None

**STAGE = 6**

**NAME = 'Identify and simplify inlined strlen() functions'**

**DESCRIPTION = 'Identify and simplify inlined strlen() functions'**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**InlinedWcscpySimplifier**

Bases: *OptimizationPass*

Simplifies inlined wide string copying logic into calls to wcsncpy, and consolidates multiple consecutive inlined wcsncpy calls.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 3**

**NAME = 'Simplify inlined wcscpy'**

**DESCRIPTION = 'Simplify inlined wcscpy patterns and consolidate multiple inlined wcsncpy calls'**

**__init__**(**args*, ***kwargs*)

**static even_offsets_are_zero**(*lst*)

**static odd_offsets_are_zero**(*lst*)

**static is_integer_likely_a_wide_string**(*v*, *size*, *endness*, *min_length=4*)

**is_inlined_wcsncpy**(*stmt*)

**class** angr.analyses.decompiler.optimization_passes.**InlinedWcscpySimplifierLate**

Bases: *InlinedWcscpySimplifier*

Same as InlinedWcscpySimplifier but runs after SSA level 1 transformation.

**STAGE = 4**

**NAME = 'Simplify inlined wcscpy (late)'**

**class** angr.analyses.decompiler.optimization_passes.**KnownPatternOutliner**

Bases: *OptimizationPass*

Automatically finds occurrences of KnownPatterns (inlined library idioms such as std::string::length() or std::vector<int>::size(); see angr.analyses.decompiler.known_patterns) in the AIL graph and outlines them into calls via the KnownPatternFinder analysis.

Runs right before variable recovery so that the synthesized calls' argument types flow into Typehoon within the same decompilation run.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 7**

**NAME = 'Outline known code patterns into calls'**

DESCRIPTION = "Automatically finds occurrences of KnownPatterns (inlined library idioms\n such as std::string::length() or std::vector<int>::size(); see\n angr.analyses.decompiler.known_patterns) in the AIL graph and outlines them\n into calls via the KnownPatternFinder analysis.\n\n Runs right before variable recovery so that the synthesized calls' argument\n types flow into Typehoon within the same decompilation run."

MAX_ROUNDS = 8

__init__(*func*, *manager*, *known_patterns=None*, *recognize_known_patterns=True*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.LoweredSwitchSimplifier

Bases: *StructuringOptimizationPass*

This optimization recognizes and reverts switch cases that have been lowered and possibly split into multiple if-else statements. This optimization, discussed in the USENIX 2024 paper SAILR, aims to undo the compiler optimization known as "Switch Lowering", present in both GCC and Clang. An in-depth discussion of this optimization can be found in the paper or in our documentation of the optimization: https://github.com/mahaloz/sailr-eval/issues/14#issue-2232616411

Note, this optimization does not occur in MSVC, which uses a different optimization strategy for switch cases. As a hack for now, we only run this deoptimization on Linux binaries.

PLATFORMS = ['linux']

NAME = 'Convert lowered switch-cases (if-else) to switch-cases'

DESCRIPTION = 'Convert lowered switch-cases (if-else) to switch-cases. Only works when the Phoenix structuring algorithm is in use.'

__init__(**args*, *min_distinct_cases=2*, ***kwargs*)

static restore_graph(*node*, *last_stmt*, *graph*, *full_graph*)

**Parameters**

- **last_stmt** (IncompleteSwitchCaseHeadStatement)
- **graph** (*DiGraph*)
- **full_graph** (*DiGraph*)

static cases_issubset(*cases_0*, *cases_1*)

Test if cases_0 is a subset of cases_1.

**Return type**
bool

**Parameters**

- **cases_0** (*list*[Case])
- **cases_1** (*list*[Case])

class angr.analyses.decompiler.optimization_passes.MipsGpSettingSimplifier

Bases: *OptimizationPass*

Removes $gp-setting statements at the beginning of MIPS functions.

ARCHES = ['MIPS32', 'MIPS64']

PLATFORMS = ['linux']

STAGE = 6

NAME = 'Remove MIPS $gp-setting statements'

DESCRIPTION = 'Removes $gp-setting statements at the beginning of MIPS functions.'

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**ModSimplifier**

Bases: *OptimizationPass*

Simplifies optimized forms of modulo computation back to "mod".

ARCHES = ['X86', 'AMD64', 'ARMCortexM', 'ARMHF', 'ARMEL']

PLATFORMS = ['linux', 'windows']

STAGE = 6

NAME = 'Simplify optimized mod forms'

DESCRIPTION = 'Simplifies optimized forms of modulo computation back to "mod".'

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**OptimizationPassStage**

Bases: Enum

Enums about optimization pass stages.

Note that the region identification pass (RegionIdentifier) may modify existing AIL blocks *without updating the topology of the original AIL graph*. For example, loop successor refinement may modify create a new AIL block with an artificial address, and alter existing jump targets of jump statements and conditional jump statements to point to this new block. However, loop successor refinement does not update the topology of the original AIL graph, which means this new AIL block does not exist in the original AIL graph. As a result, until this behavior of RegionIdentifier changes in the future, DURING_REGION_IDENTIFICATION optimization passes should not modify existing jump targets.

AFTER_AIL_GRAPH_CREATION = 0

BEFORE_SSA_LEVEL0_TRANSFORMATION = 1

AFTER_SINGLE_BLOCK_SIMPLIFICATION = 2

BEFORE_SSA_LEVEL1_TRANSFORMATION = 3

AFTER_SSA_LEVEL1_TRANSFORMATION = 4

AFTER_MAKING_CALLSITES = 5

AFTER_GLOBAL_SIMPLIFICATION = 6

BEFORE_VARIABLE_RECOVERY = 7

AFTER_VARIABLE_RECOVERY = 8

BEFORE_REGION_IDENTIFICATION = 9

DURING_REGION_IDENTIFICATION = 10

**AFTER_STRUCTURING = 11**

**class** angr.analyses.decompiler.optimization_passes.**RegisterSaveAreaSimplifier**

Bases: *OptimizationPass*

Optimizes away register spilling effects, including callee-saved registers.

This optimization runs between SSA-level0 and SSA-level1, which means registers are converted to vvars but stack accesses stay unchanged.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 2**

**NAME = 'Simplify register save areas'**

**DESCRIPTION = 'Optimizes away register spilling effects, including callee-saved registers.\n\n This optimization runs between SSA-level0 and SSA-level1, which means registers are converted to vvars but stack\n accesses stay unchanged.'**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**RegisterSaveAreaSimplifierAdvanced**

Bases: *OptimizationPass*

Optimizes away registers that are stored to or restored on the stack space.

This analysis is more complex than RegisterSaveAreaSimplifier because it handles: (1) Registers that are stored in the stack shadow space (sp+N) according to the Windows x64 calling convention. (2) Registers that are aliases of sp.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 5**

**NAME = 'Simplify register save areas (advanced)'**

**DESCRIPTION = 'Optimizes away registers that are stored to or restored on the stack space.\n\n This analysis is more complex than RegisterSaveAreaSimplifier because it handles:\n (1) Registers that are stored in the stack shadow space (sp+N) according to the Windows x64 calling convention.\n (2) Registers that are aliases of sp.'**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**RetAddrSaveSimplifier**

Bases: *OptimizationPass*

Removes code in function prologues and epilogues for saving and restoring return address registers (ra, lr, etc.), generally seen in non-leaf functions.

**ARCHES = ['MIPS32', 'MIPS64']**

**PLATFORMS = ['linux']**

**STAGE = 6**

NAME = 'Simplify return address storage'

DESCRIPTION = 'Removes code in function prologues and epilogues for saving and restoring return address registers (ra, lr, etc.),\n generally seen in non-leaf functions.'

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.ReturnDeduplicator

Bases: *OptimizationPass*

Transforms: - if (cond) { … return x; } return x;

into: - if (cond) { … } return x;

TODO: its possible that this can be expanded to all rets that are equivalent. Testing needed.

ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']

PLATFORMS = ['windows', 'linux', 'cgc']

STAGE = 10

NAME = 'Deduplicates return statements that may have been duplicated'

DESCRIPTION = 'Transforms:\n - if (cond) { ... return x; } return x;\n\n into:\n - if (cond) { ... } return x;\n\n TODO: its possible that this can be expanded to all rets that are equivalent. Testing needed.'

STRUCTURING = ['sailr', 'dream']

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHigh

Bases: *OptimizationPass*, *ReturnDuplicatorBase*

This is a light-level goto-less version of the ReturnDuplicator optimization pass. It will only duplicate return-only blocks.

ARCHES = None

PLATFORMS = None

STAGE = 6

NAME = 'Duplicate return-only blocks (high)'

DESCRIPTION = '\n This is a light-level goto-less version of the ReturnDuplicator optimization pass. It will only\n duplicate return-only blocks.\n '

STRUCTURING = ['sailr', 'dream']

__init__(**args*, *vvar_id_start*, *max_calls_in_regions=2*, *minimize_copies_for_regions=True*, *scratch=None*, ***kwargs*)

**Parameters**

- **vvar_id_start** (*int*)
- **max_calls_in_regions** (*int*)
- **minimize_copies_for_regions** (*bool*)

- **scratch** (*dict[str, Any] | None*)

class angr.analyses.decompiler.optimization_passes.**ReturnDuplicatorLow**

Bases: *StructuringOptimizationPass*, *ReturnDuplicatorBase*

An optimization pass that reverts a subset of Irreducible Statement Condensing (ISC) optimizations, as described in the USENIX 2024 paper SAILR. This is the heavy/goto version of the ReturnDuplicator optimization pass.

Some compilers, including GCC, Clang, and MSVC, apply various optimizations to reduce the number of statements in code. These optimizations will take equivalent statements, or a subset of them, and replace them with a single copy that is jumped to by gotos – optimizing for space and sometimes speed.

This optimization pass will revert those gotos by re-duplicating the condensed blocks. Since Return statements are the most common, we use this optimization pass to revert only gotos to return statements. Additionally, we perform some additional readability fixups, like not re-duplicating returns to shared components.

**Parameters**

- **func** – The function to optimize.
- **node_idx_start** – The index to start at when creating new nodes. This is used by Clinic to ensure that node indices are unique across multiple passes.
- **max_opt_iters** (int) – The maximum number of optimization iterations to perform.
- **max_calls_in_regions** (int) – The maximum number of calls that can be in a region. This is used to prevent duplicating too much code.
- **prevent_new_gotos** (bool) – If True, this optimization pass will prevent new gotos from being created.
- **minimize_copies_for_regions** (bool) – If True, this optimization pass will minimize the number of copies by doing only a single copy for connected in_edges that form a region.

**ARCHES = None**

**PLATFORMS = None**

**NAME = 'Duplicate returns connect with gotos (low)'**

**DESCRIPTION = 'An optimization pass that reverts a subset of Irreducible Statement Condensing (ISC) optimizations, as described\nin the USENIX 2024 paper SAILR. This is the heavy/goto version of the ReturnDuplicator optimization pass.\n\nSome compilers, including GCC, Clang, and MSVC, apply various optimizations to reduce the number of statements in\ncode. These optimizations will take equivalent statements, or a subset of them, and replace them with a single\ncopy that is jumped to by gotos -- optimizing for space and sometimes speed.\n\nThis optimization pass will revert those gotos by re-duplicating the condensed blocks. Since Return statements\nare the most common, we use this optimization pass to revert only gotos to return statements. Additionally, we\nperform some additional readability fixups, like not re-duplicating returns to shared components.'**

**__init__**(**args*, *max_opt_iters=4*, *max_calls_in_regions=2*, *prevent_new_gotos=True*, *minimize_copies_for_regions=True*, *region_identifier=None*, *vvar_id_start=0*, *scratch=None*, *max_func_blocks=300*, ***kwargs*)

**Parameters**

- **max_opt_iters** (*int*)
- **max_calls_in_regions** (*int*)

- **prevent_new_gotos** (*bool*)
- **minimize_copies_for_regions** (*bool*)
- **vvar_id_start** (*int*)
- **scratch** (*dict[str, Any] | None*)
- **max_func_blocks** (*int*)

class angr.analyses.decompiler.optimization_passes.**StackCanarySimplifier**

Bases: *OptimizationPass*

Removes stack canary checks from decompilation results.

**ARCHES = ['X86', 'AMD64']**

**PLATFORMS = ['cgc', 'linux']**

**STAGE = 6**

**NAME = 'Simplify stack canaries'**

**DESCRIPTION = 'Removes stack canary checks from decompilation results.'**

**__init__**(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**StaticVVarRewriter**

Bases: *OptimizationPass*

Rewrite user-specified vvars as static values or fix-sized buffers. Also rewrites reads from pointers derived off of such vvars.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 7**

**NAME = 'Static virtual variable rewriter'**

**DESCRIPTION = 'Rewrite user-specified vvars as static values or fix-sized buffers. Also rewrites reads from pointers derived off\n of such vvars.'**

**__init__**(**args*, *static_buffers=None*, *static_vvars=None*, ***kwargs*)

**Parameters**

- **static_buffers** (*dict[str, FixedBuffer] | None*)
- **static_vvars** (*dict[int, FixedBufferPtr | Const] | None*)

class angr.analyses.decompiler.optimization_passes.**SwitchDefaultCaseDuplicator**

Bases: *OptimizationPass*

For each switch-case construct (identified by jump tables), duplicate the default-case node when we detect situations where the default-case node is seemingly reused by edges outside the switch-case construct. This code reuse is usually caused by compiler code deduplication.

Ideally this pass should be implemented as an ISC optimization reversion.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 0**

**NAME = 'Duplicate default-case nodes to undo default-case node reuse caused by compiler code deduplication'**

**DESCRIPTION = 'For each switch-case construct (identified by jump tables), duplicate the default-case node when we detect\n situations where the default-case node is seemingly reused by edges outside the switch-case construct. This code\n reuse is usually caused by compiler code deduplication.\n\n Ideally this pass should be implemented as an ISC optimization reversion.'**

**__init__**(**args*, ***kwargs*)

**class** angr.analyses.decompiler.optimization_passes.**SwitchReusedEntryRewriter**

Bases: *OptimizationPass*

For each switch-case construct (identified by jump tables), rewrite the entry into a goto block when we detect situations where an entry node is reused by edges in switch-case constructs that are not the current one. This code reuse is usually caused by compiler code deduplication.

The *max_entry_reuse_count* parameter controls the maximum number of times an entry can be reused in different jump tables in the same function; exceeding this threshold probably indicates that the jump table recovery is incorrect and thus creating jump tables with too many duplicate entries.

The *max_reused_entries* parameter controls the maximum number of reused entries that can be found in a function before aborting this optimization.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 0**

**NAME = 'Rewrite switch-case entry nodes with multiple predecessors into goto statements.'**

**DESCRIPTION = 'For each switch-case construct (identified by jump tables), rewrite the entry into a goto block when we detect\n situations where an entry node is reused by edges in switch-case constructs that are not the current one. This code\n reuse is usually caused by compiler code deduplication.\n\n The `max_entry_reuse_count` parameter controls the maximum number of times an entry can be reused in different jump\n tables in the same function; exceeding this threshold probably indicates that the jump table recovery is incorrect\n and thus creating jump tables with too many duplicate entries.\n\n The `max_reused_entries` parameter controls the maximum number of reused entries that can be found in a function\n before aborting this optimization.'**

**__init__**(**args*, *max_entry_reuse_count=10*, *max_reused_entries=20*, ***kwargs*)

**Parameters**

- **max_entry_reuse_count** (*int*)
- **max_reused_entries** (*int*)

class angr.analyses.decompiler.optimization_passes.**TagSlicer**

Bases: *OptimizationPass*

Removes unmarked statements from the graph.

**ARCHES = None**

**PLATFORMS = None**

**STAGE = 8**

**NAME = 'Remove unmarked statements from the graph.'**

**DESCRIPTION = 'Removes unmarked statements from the graph.'**

**__init__**(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**WinStackCanarySimplifier**

Bases: *OptimizationPass*

Removes stack canary checks from decompilation results for Windows PE files.

we need to run this pass before performing any full-function simplification. Otherwise the effects of _security_cookie will be propagated.

**ARCHES = ['X86', 'AMD64']**

**PLATFORMS = ['windows']**

**STAGE = 2**

**NAME = 'Simplify stack canaries in Windows PE files'**

**DESCRIPTION = 'Removes stack canary checks from decompilation results for Windows PE files.\n\n we need to run this pass before performing any full-function simplification. Otherwise the effects of\n _security_cookie will be propagated.'**

**__init__**(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.**X86GccGetPcSimplifier**

Bases: *OptimizationPass*

Simplifies __x86.get_pc_thunk calls.

**ARCHES = ['X86']**

**PLATFORMS = ['linux']**

**STAGE = 1**

**NAME = 'Simplify getpc()'**

**DESCRIPTION = 'Simplifies __x86.get_pc_thunk calls.'**

**__init__**(**args*, ***kwargs*)

angr.analyses.decompiler.optimization_passes.**get_optimization_passes**(*arch*, *platform*)

angr.analyses.decompiler.optimization_passes.**register_optimization_pass**(*opt_pass*, ***, *presets=None*)

**Parameters**

**presets** (*list[str | DecompilationPreset] | None*)

### Submodules


*base_ptr_save_simplifier*
*call_stmt_rewriter*
*code_motion*
*condition_constprop*
*const_derefs*
*const_prop_reverter*
*cross_jump_reverter*
*deadblock_remover*
*determine_load_sizes*
*div_simplifier*
*duplication_reverter*
*eager_std_string_concatenation*
*eager_std_string_eval*
*engine_base*
*expr_op_swapper*
*flip_boolean_cmp*
*inlined_memcpy_simplifier*
*inlined_memset_simplifier*
*inlined_strcpy_simplifier*
*inlined_string_transformation_simplifier*
*inlined_strlen_simplifier*
*inlined_wcscpy_simplifier*
*ite_expr_converter*
*ite_region_converter*
*known_pattern_outliner*
*lowered_switch_simplifier*
*mips_gp_setting_simplifier*
*mod_simplifier*
*optimization_pass*
*peephole_simplifier*
*register_save_area_simplifier*
*register_save_area_simplifier_adv*
*ret_addr_save_simplifier*
*ret_deduplicator*
*return_duplicator_base*
*return_duplicator_high*
*return_duplicator_low*
*stack_canary_simplifier*
*static_vvar_rewriter*
*switch_default_case_duplicator*
*switch_reused_entry_rewriter*
*tag_slicer*
*win_stack_canary_simplifier*
*x86_gcc_getpc_simplifier*


#### angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier

**class** angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.**BasePointerSaveSimplifier**

Bases: *OptimizationPass*

Removes the effects of base pointer stack storage at function invocation and restoring at function return.

**ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 6**

**NAME = 'Simplify base pointer saving'**

**DESCRIPTION = 'Removes the effects of base pointer stack storage at function invocation and restoring at function return.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.call_stmt_rewriter

**class** angr.analyses.decompiler.optimization_passes.call_stmt_rewriter.**CallStatementRewriter**

Bases: *OptimizationPass*

Rewrite call statements to assignments if needed.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 5**

**NAME = 'Unify call statements on demand.'**

**DESCRIPTION = 'Rewrite call statements to assignments if needed.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.code_motion

**class** angr.analyses.decompiler.optimization_passes.code_motion.**CodeMotionOptimization**

Bases: *OptimizationPass*

Moves common statements out of blocks that share the same predecessors or the same successors. This is done to reduce the number of statements in a block and to make the blocks more similar to each other.

As an example:

```
if (x) {
    b = 2;
    a = 1;
    c = 3;
} else {
    b = 2;
    c = 3;
}
```

Will be turned into:

```
if (x) {
    a = 1;
}
b = 2;
c = 3;
```

Current limitations (for very conservative operations):

- moving statements above conditional jumps is not supported
- only immediate children and parents are considered for moving statements
- when moving statements down, a block is only considered if already has a matching statement at the end

**ARCHES = None**

**PLATFORMS = None**

**NAME = 'Merge common statements in sub-scopes'**

**STAGE: *OptimizationPassStage* = 6**

**DESCRIPTION = '\n Moves common statements out of blocks that share the same predecessors or the same\n successors. This is done to reduce the number of statements in a block and to make the\n blocks more similar to each other.\n\n As an example::\n\n if (x) {\n b = 2;\n a = 1;\n c = 3;\n } else {\n b = 2;\n c = 3;\n }\n\n Will be turned into::\n\n if (x) {\n a = 1;\n }\n b = 2;\n c = 3;\n\n Current limitations (for very conservative operations):\n\n - moving statements above conditional jumps is not supported\n - only immediate children and parents are considered for moving statements\n - when moving statements down, a block is only considered if already has a matching statement at the end\n '**

**__init__**(**args*, *max_iters=10*, *node_idx_start=0*, ***kwargs*)

**Parameters**
**node_idx_start** (*int*)

**static update_graph_with_super_edits**(*original_graph*, *super_graph*, *updated_blocks*)

This function updates an graph when doing block edits on a supergraph version of that same graph. The updated blocks must be provided as a dictionary where the keys are original block in the supergraph and the values are the new blocks that should replace them.

The supergraph MUST be generated using the to_ail_supergraph function, since it stores the original nodes each super node represents. This is necessary to update the original graph with the new super nodes.

**Return type**
bool

**Parameters**

- **original_graph** (*DiGraph*)
- **super_graph** (*DiGraph*)
- **updated_blocks** (*dict[Block, Block]*)

### angr.analyses.decompiler.optimization_passes.condition_constprop

**class** angr.analyses.decompiler.optimization_passes.condition_constprop.**ConstantCondition**

Bases: object

Describes an opportunity for replacing a vvar with a constant value.

**__init__**(*vvar_id*, *value*, *block_addr*, *block_idx*)

**Parameters**

- **vvar_id** (*int*)
- **value** (Const)
- **block_addr** (*int*)
- **block_idx** (*int | None*)

**class** angr.analyses.decompiler.optimization_passes.condition_constprop.**CCondPropBlockWalker**

Bases: *AILBlockRewriter*

Block walker for ConditionConstantPropagation to replace vvars with constant values.

**__init__**(*vvar_id*, *const_value*)

**Parameters**

- **vvar_id** (*int*)
- **const_value** (Const)

**class** angr.analyses.decompiler.optimization_passes.condition_constprop.**ConditionConstantPropagation**

Bases: *OptimizationPass*

Reason about constant propagation opportunities from conditionals and propagate constants in the graph accordingly.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 2**

**NAME = 'Propagate constants using information deduced from conditionals.'**

**DESCRIPTION = 'Reason about constant propagation opportunities from conditionals and propagate constants in the graph accordingly.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.const_derefs

**class** angr.analyses.decompiler.optimization_passes.const_derefs.**BlockWalker**

Bases: *AILBlockRewriter*

AIL Block walker in order to perform const deref substitution

__init__(*project*, *manager*)

**Parameters**

- **project** (Project)
- **manager** (*Manager*)

**class** angr.analyses.decompiler.optimization_passes.const_derefs.**ConstantDereferencesSimplifier**

Bases: *OptimizationPass*

Makes the following simplifications:

```
*(*(const_addr))  ==>  *(value) iff  *const_addr == value
```

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 2**

**NAME = 'Simplify constant dereferences'**

**DESCRIPTION = 'Makes the following simplifications::\n\n *(*(const_addr)) ==>
*(value) iff *const_addr == value'**

__init__(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.const_prop_reverter

**class** angr.analyses.decompiler.optimization_passes.const_prop_reverter.**BlockWalker**

Bases: *AILBlockViewer*

__init__(*walked_objs*)

**Parameters**

**walked_objs** (*dict[*type, *set[*SideEffectStatement | Assignment | Return | Store*]]*)

**class** angr.analyses.decompiler.optimization_passes.const_prop_reverter.**ConstPropOptReverter**

Bases: *OptimizationPass*

This optimization reverts the effects of constant propagation done by the compiler as discussed in the USENIX 2024 paper SAILR. This optimization's main goal is to enable later optimizations that rely on symbolic variables to be more effective. This optimization pass will convert two statements with a difference of a const and a symbolic variable into two statements with the symbolic variables.

As an example: x = 75 puts(x) puts(75)

will be converted to: x = 75 puts(x) puts(x)

**ARCHES = None**

**PLATFORMS = None**

**STRUCTURING: list[str] | None = ['sailr', 'dream']**

**STAGE: *OptimizationPassStage* = 10**

NAME = 'Revert Constant Propagation Optimizations'

DESCRIPTION = "This optimization reverts the effects of constant propagation done by the compiler as discussed in the\n USENIX 2024 paper SAILR. This optimization's main goal is to enable later optimizations that rely on\n symbolic variables to be more effective. This optimization pass will convert two statements with a difference of\n a const and a symbolic variable into two statements with the symbolic variables.\n\n As an example:\n x = 75\n puts(x)\n puts(75)\n\n will be converted to:\n x = 75\n puts(x)\n puts(x)"

__init__(**args*, *region_identifier=None*, *reaching_definitions=None*, ***kwargs*)

static find_conflicting_call_args(*call0*, *call1*)

**Parameters**

- **call0** (Call)
- **call1** (Call)

### angr.analyses.decompiler.optimization_passes.cross_jump_reverter

class angr.analyses.decompiler.optimization_passes.cross_jump_reverter.**CrossJumpReverter**

Bases: *StructuringOptimizationPass*

This is an implementation to revert the compiler optimization Cross Jumping, an ISC optimization discussed in the USENIX 2024 paper SAILR. This optimization is somewhat aggressive and as such should be run last in your decompiler deoptimization chain. This deoptimization will take any goto it finds and attempt to duplicate its target block if its target only has one outgoing edge.

There are some heuristics in place to prevent duplication everywhere. First, this deoptimization will only run a max of max_opt_iters times. Second, it will not duplicate a block with too many calls.

STAGE: *OptimizationPassStage* = 10

NAME = 'Duplicate linear blocks with gotos'

DESCRIPTION = 'This is an implementation to revert the compiler optimization Cross Jumping, an ISC optimization discussed\nin the USENIX 2024 paper SAILR. This optimization is somewhat aggressive and as such should be run last in your\ndecompiler deoptimization chain. This deoptimization will take any goto it finds and attempt to duplicate its\ntarget block if its target only has one outgoing edge.\n\nThere are some heuristics in place to prevent duplication everywhere. First, this deoptimization will only run\na max of max_opt_iters times. Second, it will not duplicate a block with too many calls.'

__init__(**args*, *node_idx_start=0*, *max_opt_iters=3*, *max_call_duplications=1*, ***kwargs*)

**Parameters**

- **node_idx_start** (*int*)
- **max_opt_iters** (*int*)
- **max_call_duplications** (*int*)

### angr.analyses.decompiler.optimization_passes.deadblock_remover

class angr.analyses.decompiler.optimization_passes.deadblock_remover.**DeadblockRemover**

Bases: *OptimizationPass*

Removes condition-unreachable blocks from the graph.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 9**

**NAME = 'Remove blocks with unsatisfiable conditions'**

**DESCRIPTION = 'Removes condition-unreachable blocks from the graph.'**

**__init__**(**args*, *node_cutoff=200*, ***kwargs*)

**Parameters**
**node_cutoff** (*int*)

### angr.analyses.decompiler.optimization_passes.determine_load_sizes

class angr.analyses.decompiler.optimization_passes.determine_load_sizes.**LoadSizeRewriter**

Bases: *AILBlockRewriter*

Rewrite Load expressions whose sizes are undetermined into loads of the strings they point to.

**__init__**(*project*)

**Parameters**
**project** (Project)

class angr.analyses.decompiler.optimization_passes.determine_load_sizes.**DetermineLoadSizes**

Bases: *OptimizationPass*

Determine the sizes of Load expressions whose sizes are undetermined.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 6**

**NAME = 'Determine sizes of loads whose sizes are undetermined'**

**DESCRIPTION = 'Determine the sizes of Load expressions whose sizes are undetermined.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.div_simplifier

class angr.analyses.decompiler.optimization_passes.div_simplifier.**DivSimplifierAILEngine**

Bases: *SimplifierAILEngine*

An AIL pass for the div simplifier

__init__(*project*, *manager*)

class angr.analyses.decompiler.optimization_passes.div_simplifier.**DivSimplifier**

Bases: *OptimizationPass*

Simplifies various division optimizations back to "div".

**ARCHES = ['X86', 'AMD64', 'ARMCortexM', 'ARMHF', 'ARMEL']**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 6**

**NAME = 'Simplify arithmetic division'**

**DESCRIPTION = 'Simplifies various division optimizations back to "div".'**

__init__(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.duplication_reverter

class
angr.analyses.decompiler.optimization_passes.duplication_reverter.**DuplicationReverter**

Bases: *StructuringOptimizationPass*

This (de)optimization reverts the effects of many compiler optimizations that cause code duplication in the decompilation. This deoptimization is the implementation of the USENIX 2024 paper SAILR's ISD doptimization. As such, the main goal of this optimization is to remove code duplication by merging semantically similar blocks in the AIL graph.

**NAME = 'Revert Statement Duplication Optimizations'**

**DESCRIPTION = "This (de)optimization reverts the effects of many compiler optimizations that cause code duplication in\n the decompilation. This deoptimization is the implementation of the USENIX 2024 paper SAILR's ISD\n doptimization. As such, the main goal of this optimization is to remove code duplication by merging\n semantically similar blocks in the AIL graph."**

__init__(**args*, *max_guarding_conditions=4*, ***kwargs*)

**write_graph: DiGraph | None**

**read_graph: DiGraph | None**

**static boolean_operators_in_condition**(*condition*)

TODO: this entire boolean checking semantic we use needs to be removed, see how it is used for other dels needed we need to replace it with a boolean variable insertion on both branches that lead to the new block.

Say we have:

```
if (A()) {
    do_thing();
}
if (B()) {
    do_thing():
}
```

We want to translate it to:

```
int should_do_thing = 0;
if (A())
    should_do_thing = 1;
if (B())
    should_do_thing = 1;

if (should_do_thing):
    do_thing();
```

Although longer, this code can be optimized to look like:

```
int should_do_thing = A() || B();
if (should_do_thing)
    do_thing();
```

**Parameters**
**condition** (Expression)

**stmt_can_move_to**(*stmt*, *block*, *new_idx*, *io_finder=None*)

**maximize_similarity_of_blocks**(*block1*, *block2*, *graph*)

This attempts to rearrange the order of statements in block1 and block2 to maximize the similarity between them. This implementation is a little outdated since CodeMotion optimization was implemented, but it should be disabled until we have a good SSA implementation.

TODO: reimplement me when we have better SSA

**Return type**
tuple[Block, Block]

**create_merged_subgraph**(*blocks*, *graph*, *maximize_similarity=False*)

**Return type**
*AILMergeGraph*

**Parameters**
**graph** (*DiGraph*)

**similar_conditional_when_single_corrected**(*block1*, *block2*, *graph*)

**Parameters**

- **block1** (*Block*)
- **block2** (*Block*)
- **graph** (*DiGraph*)

**collect_conditions_between_nodes**(*graph*, *source*, *sinks*, *max_depth=15*)

**Parameters**

- **source** (*Block*)
- **sinks** (*list[Block]*)

**shared_common_conditional_dom**(*nodes*, *graph*)

Takes n nodes and returns True only if all the nodes are dominated by the same node, which must be a ConditionalJump

@param nodes: @param graph: @return:

**Parameters**
**graph** (*DiGraph*)

## Submodules


*ail_merge_graph*
*duplication_reverter*
*errors*
*similarity*
*utils*


### angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph

**class** angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph.**AILBlockSplit**

Bases: object

This class represents a block that has been split into three parts, which is best explained in the AILMergeGraph class example. See that class for more information.

The up_split, is all statements above the matched Longest Common Sequence (LCS), the match_split is the LCS, and the down_split is all statements below the matched LCS. This class should only be used in the context of the AILMergeGraph class.

**__init__**(*original=None*, *up_split=None*, *match_split=None*, *down_split=None*)

**Parameters**

- **original**
- **up_split** – The block split above the matched LCS
- **match_split** – The block containing only the matched LCS
- **down_split** – The block split below the matched LCS

**classmethod from_block_lcs**(*original_block*, *idx*, *len_*)

**Parameters**
**original_block** (*Block*)

**static split_ail_block**(*block*, *split_idx*, *split_len*)

**Return type**
tuple[Block | None, Block | None, Block | None]

**class** angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph.**AILMergeGraph**

Bases: object

This class represents the merged results of two AIL graphs that have been found to be similar. We can reference these graphs as G1 and G2. The two graphs are of the following form, where any node other than D can be empty:

```
  A
 /          B   C
\  /
```

(continues on next page)

(continued from previous page)

```
 D
/          E   F
```

The D node can be a subgraph, but in both G1 and G2, this D-subgraph are exact duplicates of each other, except their top and bottom statements. This class is the result of merging those two D subgraphs.

To explain that last part about statements differing at the ends, see this example:

```
D1:
-----
a = 10;
puts(a);
puts("bye");
-----

D2:
-----
a = 11;
puts(a);
puts("cya");
-----
```

In this case, the merged D would contain just `puts(a)`. The statements above it, referred to as up_split in the code, and the statements below it, referred to as down_split in the code, would be moved out of the block and bounded by the conditions that lead to those statements. This creates a graph even in the case of a single block being the original D.

Lastly, since this class will deal a lot with splitting blocks into pieces, we keep a mapping of how the original blocks turned into the new ones and vice versa.

**__init__**(*graph=None*, *original_graph=None*, *conditional_block=None*, *original_blocks=None*, *original_split_blocks=None*)

**create_conditionless_graph**(*starting_blocks*, *graph_lcs*)

**Return type**
dict[Block, Block] | None

**Parameters**
**starting_blocks** (*list[Block]*)

**add_edges_to_condition**(*conditional_block*, *true_target*, *merge_end_pairs*)

**create_mapping_to_merge_graph**(*updated_blocks*, *start_blocks*)

**Parameters**

- **updated_blocks** (*dict[Block, Block]*)
- **start_blocks** (*list[Block]*)

**merged_is_split_type**(*merge_block*, *split_type*)

**Parameters**

- **merge_block** (*Block*)
- **split_type** (*str*)

**static clone_graph_replace_splits**(*graph*, *split_map*)

**Parameters**

- **graph** (*DiGraph*)
- **split_map** (*dict[Block, Block]*)

### angr.analyses.decompiler.optimization_passes.duplication_reverter.duplication_reverter

**class** angr.analyses.decompiler.optimization_passes.duplication_reverter.duplication_reverter.**DuplicationReverter**

Bases: *StructuringOptimizationPass*

This (de)optimization reverts the effects of many compiler optimizations that cause code duplication in the decompilation. This deoptimization is the implementation of the USENIX 2024 paper SAILR's ISD doptimization. As such, the main goal of this optimization is to remove code duplication by merging semantically similar blocks in the AIL graph.

**NAME = 'Revert Statement Duplication Optimizations'**

**DESCRIPTION = "This (de)optimization reverts the effects of many compiler optimizations that cause code duplication in\n the decompilation. This deoptimization is the implementation of the USENIX 2024 paper SAILR's ISD\n doptimization. As such, the main goal of this optimization is to remove code duplication by merging\n semantically similar blocks in the AIL graph."**

**__init__**(**args*, *max_guarding_conditions=4*, ***kwargs*)

**write_graph: DiGraph | None**

**read_graph: DiGraph | None**

**static boolean_operators_in_condition**(*condition*)

TODO: this entire boolean checking semantic we use needs to be removed, see how it is used for other dels needed we need to replace it with a boolean variable insertion on both branches that lead to the new block.

Say we have:

```
if (A()) {
    do_thing();
}
if (B()) {
    do_thing():
}
```

We want to translate it to:

```
int should_do_thing = 0;
if (A())
    should_do_thing = 1;
if (B())
    should_do_thing = 1;

if (should_do_thing):
    do_thing();
```

Although longer, this code can be optimized to look like:

```
int should_do_thing = A() || B();
if (should_do_thing)
    do_thing();
```

**Parameters**
**condition** (Expression)

**stmt_can_move_to**(*stmt*, *block*, *new_idx*, *io_finder=None*)

**maximize_similarity_of_blocks**(*block1*, *block2*, *graph*)

This attempts to rearrange the order of statements in block1 and block2 to maximize the similarity between them. This implementation is a little outdated since CodeMotion optimization was implemented, but it should be disabled until we have a good SSA implementation.

TODO: reimplement me when we have better SSA

**Return type**
tuple[Block, Block]

**create_merged_subgraph**(*blocks*, *graph*, *maximize_similarity=False*)

**Return type**
*AILMergeGraph*

**Parameters**
**graph** (*DiGraph*)

**similar_conditional_when_single_corrected**(*block1*, *block2*, *graph*)

**Parameters**

- **block1** (*Block*)
- **block2** (*Block*)
- **graph** (*DiGraph*)

**collect_conditions_between_nodes**(*graph*, *source*, *sinks*, *max_depth=15*)

**Parameters**

- **source** (*Block*)
- **sinks** (*list[Block]*)

**shared_common_conditional_dom**(*nodes*, *graph*)

Takes n nodes and returns True only if all the nodes are dominated by the same node, which must be a ConditionalJump

@param nodes: @param graph: @return:

**Parameters**
**graph** (*DiGraph*)

### angr.analyses.decompiler.optimization_passes.duplication_reverter.errors

**exception** angr.analyses.decompiler.optimization_passes.duplication_reverter.errors.**SAILRSemanticError**

Bases: Exception

These types of errors may not kill the entire analysis, but they do kill the current working round. They are caused when some non-expected state happens, like a ConditionalJump that is not the last statement in a block.

**exception** angr.analyses.decompiler.optimization_passes.duplication_reverter.errors.**UnsupportedAILNodeError**

Bases: *SAILRSemanticError*

These type of errors are raised when an AIL node is not supported by the current implementation. Most common with Switch heads.

### angr.analyses.decompiler.optimization_passes.duplication_reverter.similarity

angr.analyses.decompiler.optimization_passes.duplication_reverter.similarity.**longest_ail_graph_subseq**(*bl*
*gr*

angr.analyses.decompiler.optimization_passes.duplication_reverter.similarity.**ail_graph_similarity**(*block0*,
*block1*,
*graph*,
*only_b*

**Parameters**

- **block0** (*Block*)
- **block1** (*Block*)
- **graph** (*DiGraph*)

angr.analyses.decompiler.optimization_passes.duplication_reverter.similarity.**ail_similarity_to_orig_bloc**

angr.analyses.decompiler.optimization_passes.duplication_reverter.similarity.**find_block_by_similarity**(*bl*
*gr*
*no*

### angr.analyses.decompiler.optimization_passes.duplication_reverter.utils

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**find_block_in_successors_by_addr**

**Return type**
Block | None

**Parameters**

- **addr** (*int*)
- **block** (*Block*)
- **graph** (*DiGraph*)

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**replace_node_in_graph**(*graph*,
*node*,
*re-*
*place_with*)

**Parameters**
**graph** (*DiGraph*)

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**bfs_list_blocks**(*start_block*, *graph*)

**Parameters**

- **start_block** (*Block*)
- **graph** (*DiGraph*)

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**copy_graph_and_nodes**(*graph*)

**Parameters**
**graph** (*DiGraph*)

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**ail_block_from_stmts**(*stmts*, *idx=None*, *block_addr=*

**Return type**
Block | None

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**deepcopy_ail_jump**(*stmt*, *idx=1*)

**Parameters**
**stmt** (Jump)

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**deepcopy_ail_condjump**(*stmt*, *idx=1*)

**Parameters**
**stmt** (ConditionalJump)

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**deepcopy_ail_anyjump**(*stmt*, *idx=1*)

**Parameters**
**stmt** (Jump | ConditionalJump)

angr.analyses.decompiler.optimization_passes.duplication_reverter.utils.**correct_jump_targets**(*stmt*, *replacement_map*, *new_stmt=Tr*

**Parameters**
**replacement_map** (*dict[int, int]*)

### angr.analyses.decompiler.optimization_passes.eager_std_string_concatenation

**class** angr.analyses.decompiler.optimization_passes.eager_std_string_concatenation.**EagerStdStringConcatenationPass**

Bases: *OptimizationPass*

Concatenate multiple constant std::string creation calls into one when possible.

**ARCHES = None**

PLATFORMS = None

STAGE: *OptimizationPassStage* = 7

NAME = 'Condense multiple constant std::string creation calls into one when possible'

DESCRIPTION = 'Concatenate multiple constant std::string creation calls into one when possible.'

__init__(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.eager_std_string_eval

class angr.analyses.decompiler.optimization_passes.eager_std_string_eval.**RewriteStdStringCallWalker**

Bases: *AILBlockRewriter*

__init__(*str_defs*, *kb*, *manager*, ***kwargs*)

**Parameters**

- **str_defs** (*dict[int, bytes]*)
- **manager** (*Manager*)

class angr.analyses.decompiler.optimization_passes.eager_std_string_eval.**EagerStdStringEvalPass**

Bases: *OptimizationPass*

Eagerly evaluate std::string methods for constant std::string instances.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 7

NAME = 'Eagerly evaluate std::string methods for constant std::string instances'

DESCRIPTION = 'Eagerly evaluate std::string methods for constant std::string instances.'

__init__(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.engine_base

class angr.analyses.decompiler.optimization_passes.engine_base.**SimplifierAILState**

Bases: object

The abstract state used in SimplifierAILEngine.

__init__(*arch*, *variables=None*)

copy()

merge(**others*)

**store_variable**(*old*, *new*)

**Parameters**
**old** (VirtualVariable)

**get_variable**(*old*)

**Parameters**
**old** (VirtualVariable)

**remove_variable**(*old*)

**class** angr.analyses.decompiler.optimization_passes.engine_base.**SimplifierAILEngine**

Bases: *SimEngineLightAIL*[*SimplifierAILState*, *Expression*, *Statement*, Block]

Essentially implements a peephole optimization engine for AIL statements (because we do not perform memory or register loads).

### angr.analyses.decompiler.optimization_passes.expr_op_swapper

**class** angr.analyses.decompiler.optimization_passes.expr_op_swapper.**OuterWalker**

Bases: *SequenceWalker*

A sequence walker that finds nodes and invokes expression replacer to replace expressions.

**__init__**(*desc*)

**desc:** dict[*OpDescriptor*, str]

**class** angr.analyses.decompiler.optimization_passes.expr_op_swapper.**ExpressionReplacer**

Bases: *AILBlockRewriter*

Replace expressions.

**__init__**(*block_addr*, *target_expr_predicate*, *callback*)

**class** angr.analyses.decompiler.optimization_passes.expr_op_swapper.**OpDescriptor**

Bases: object

Describes a specific operator. Serializes to JSON.

**__init__**(*block_addr*, *stmt_idx*, *ins_addr*, *op*)

**Parameters**

- **block_addr** (*int*)
- **stmt_idx** (*int*)
- **ins_addr** (*int*)
- **op** (*str*)

**to_jsonable**()

**Return type**
dict[str, Any]

**to_json**()

**Return type**
str

classmethod **from_jsonable**(*d*)

**Return type**
*OpDescriptor*

**Parameters**
**d** (*dict[str, Any]*)

classmethod **from_json**(*s*)

**Return type**
*OpDescriptor*

**Parameters**
**s** (*str*)

class angr.analyses.decompiler.optimization_passes.expr_op_swapper.**ExprOpSwapper**

Bases: *SequenceOptimizationPass*

Swap operands (and the operator accordingly) in a BinOp expression.

**ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']**

**PLATFORMS = ['windows', 'linux', 'cgc']**

**STAGE: *OptimizationPassStage* = 11**

**NAME = 'Swap operands of expressions as requested'**

**DESCRIPTION = 'Swap operands (and the operator accordingly) in a BinOp expression.'**

**__init__**(**args*, *binop_operators=None*, ***kwargs*)

**Parameters**
**binop_operators** (*dict[OpDescriptor, str] | None*)

### angr.analyses.decompiler.optimization_passes.flip_boolean_cmp

class angr.analyses.decompiler.optimization_passes.flip_boolean_cmp.**FlipBooleanWalker**

Bases: *SequenceWalker*

Walks a SequenceNode and handles every sequence. Uses the flip_size to determine when to flip the condition on large if-statement bodies.

**__init__**(*graph*, *manager*, *flip_size=9*, *last_node=None*)

**Parameters**
**manager** (*Manager*)

class angr.analyses.decompiler.optimization_passes.flip_boolean_cmp.**FlipBooleanCmp**

Bases: *SequenceOptimizationPass*

In the scenario in which a false node has no apparent successors, flip the condition on that if-stmt. This is only useful when StructuredCodeGenerator has simplify_else_scopes enabled, as this will allow the flipped if-stmt to remove the redundant else.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 11**

NAME = 'Flip small ret booleans'

DESCRIPTION = 'When false node has no successors, flip condition so else scope can be simplified later'

__init__(*args*, *flip_size=9*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.inlined_memcpy_simplifier

class angr.analyses.decompiler.optimization_passes.inlined_memcpy_simplifier.InlinedMemcpySimplifier

Bases: *OptimizationPass*

Simplifies inlined data copying logic into calls to memcpy.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 3

NAME = 'Simplify inlined memcpy'

DESCRIPTION = 'Simplify inlined memcpy patterns into memcpy calls'

__init__(*args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.inlined_memcpy_simplifier.InlinedMemcpySimplifierLate

Bases: *InlinedMemcpySimplifier*

Same as InlinedMemcpySimplifier but runs after SSA level 1 transformation.

STAGE: *OptimizationPassStage* = 4

NAME = 'Simplify inlined memcpy (late)'

### angr.analyses.decompiler.optimization_passes.inlined_memset_simplifier

class angr.analyses.decompiler.optimization_passes.inlined_memset_simplifier.ConstAssignmentInfo

Bases: object

Holds information about a constant assignment.

__init__(*kind*, *base*, *offset*, *value*, *bits*, *count=1*)

**Parameters**

- **kind** (*Literal['stack', 'heap', 'global']*)
- **base** (*Any*)
- **offset** (*int*)
- **value** (*int*)
- **bits** (*int*)
- **count** (*int*)

kind: Literal['stack', 'heap', 'global']

base

offset

value

bits

count

class angr.analyses.decompiler.optimization_passes.inlined_memset_simplifier.InlinedMemsetSimplifier

Bases: *OptimizationPass*

Simplifies inlined memory setting logic into calls to memset.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 3

NAME = 'Simplify inlined memset'

DESCRIPTION = 'Simplify inlined memset patterns into memset calls'

MIN_ASSIGNMENTS = 2

__init__(**args*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.inlined_memset_simplifier.InlinedMemsetSimplifierLate

Bases: *InlinedMemsetSimplifier*

Same as InlinedMemsetSimplifier but runs after SSA level 1 transformation.

STAGE: *OptimizationPassStage* = 4

NAME = 'Simplify inlined memset (late)'

### angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier

class angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier.InlinedStrcpySimplifier

Bases: *OptimizationPass*

Simplifies inlined string copying logic into calls to strcpy/strncpy, and consolidates multiple consecutive inlined strcpy calls.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 3

NAME = 'Simplify inlined strcpy'

**DESCRIPTION = 'Simplify inlined strcpy patterns and consolidate multiple inlined strcpy calls'**

**__init__**(**args*, ***kwargs*)

**static is_integer_likely_a_string**(*v*, *size*, *endness*, *min_length=4*)

**is_inlined_strcpy**(*stmt*)

**class** angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier.**InlinedStrcpySimplifierLate**

Bases: *InlinedStrcpySimplifier*

Same as InlinedStrcpySimplifier but runs after SSA level 1 transformation.

**STAGE: *OptimizationPassStage* = 4**

**NAME = 'Simplify inlined strcpy (late)'**

### angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier

**class** angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.**FasterMemory**

Bases: *SimpleInterfaceMixin*, *DefaultFillerMixin*, *UltraPagesMixin*, *PagedMemoryMixin*

A fast memory model used in InlinedStringTransformationState.

**class** angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.**InlinedStringTransformationState**

Bases: object

The abstract state used in InlinedStringTransformationAILEngine.

**__init__**(*project*)

**reg_store**(*reg*, *value*)

**Return type**
None

**Parameters**

- **reg** (Register)
- **value** (*BV*)

**reg_load**(*reg*)

**Return type**
BV | None

**Parameters**
**reg** (Register)

**mem_store**(*addr*, *value*, *endness*)

**Return type**
None

**Parameters**

- **addr** (*int*)

- **value** (*Bits*)
- **endness** (*str*)

**mem_load**(*addr*, *size*, *endness*)

**Return type**
BV | None

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**vvar_store**(*vvar*, *value*)

**Return type**
None

**Parameters**

- **vvar** (VirtualVariable)
- **value** (*Bits | None*)

**vvar_load**(*vvar*)

**Return type**
BV | None

**Parameters**
**vvar** (VirtualVariable)

**class** angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.**InlinedStringTransformationAILEngine**

Bases: *SimEngineNostmtAIL*[*InlinedStringTransformationState*, BV | None, None, None]

A simple AIL execution engine

**__init__**(*project*, *nodes*, *start*, *end*, *step_limit*)

**Parameters**

- **nodes** (*dict[int, Any]*)
- **start** (*int*)
- **end** (*int*)
- **step_limit** (*int*)

**nodes:** dict[int, Any]

**start:** int

**end:** int

**step_limit:** int

**stack_accesses:** defaultdict[int, list[tuple[str, *CodeLocation*, Bits]]]

**finished:** bool

class angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.**InlineStringTransformationDescriptor**

Bases: object

Describes an instance of inline string transformation.

**__init__**(*store_block*, *loop_body*, *stack_accesses*, *beginning_stack_offset*)

class angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.**InlinedStringTransformationSimplifier**

Bases: *OptimizationPass*

Simplifies inlined string transformation routines.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 3**

**NAME = 'Simplify string transformations'**

**DESCRIPTION = 'Simplify string transformations that are commonly used in obfuscated functions.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.inlined_strlen_simplifier

class angr.analyses.decompiler.optimization_passes.inlined_strlen_simplifier.**InlinedStrlenSimplifier**

Bases: *OptimizationPass*

Abstracts inlined strlen functions into strlen() calls.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 6**

**NAME = 'Identify and simplify inlined strlen() functions'**

**DESCRIPTION = 'Identify and simplify inlined strlen() functions'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier

class angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier.**InlinedWcscpySimplifier**

Bases: *OptimizationPass*

Simplifies inlined wide string copying logic into calls to wcsncpy, and consolidates multiple consecutive inlined wcsncpy calls.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 3**

**NAME = 'Simplify inlined wcscpy'**

**DESCRIPTION = 'Simplify inlined wcscpy patterns and consolidate multiple inlined wcsncpy calls'**

**__init__**(**args*, ***kwargs*)

**static even_offsets_are_zero**(*lst*)

**static odd_offsets_are_zero**(*lst*)

**static is_integer_likely_a_wide_string**(*v*, *size*, *endness*, *min_length=4*)

**is_inlined_wcsncpy**(*stmt*)

**class** angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier.**InlinedWcscpySimplifierLate**

Bases: *InlinedWcscpySimplifier*

Same as InlinedWcscpySimplifier but runs after SSA level 1 transformation.

**STAGE: *OptimizationPassStage* = 4**

**NAME = 'Simplify inlined wcscpy (late)'**

### angr.analyses.decompiler.optimization_passes.ite_expr_converter

**exception** angr.analyses.decompiler.optimization_passes.ite_expr_converter.**NodeFoundNotification**

Bases: Exception

A notification that the target node has been found.

**class** angr.analyses.decompiler.optimization_passes.ite_expr_converter.**BlockLocator**

Bases: *RegionWalker*

Recursively locate block in a RegionOverlay instance.

It might be reasonable to move this class into its own file.

**__init__**(*block*)

**walk_node**(*region*, *node*)

**class** angr.analyses.decompiler.optimization_passes.ite_expr_converter.**ExpressionReplacer**

Bases: *AILBlockRewriter*

Replace expressions.

**__init__**(*block_addr*, *target_expr*, *callback*)

**class** angr.analyses.decompiler.optimization_passes.ite_expr_converter.**ITEExprConverter**

Bases: *OptimizationPass*

Transform specific expressions into If-Then-Else expressions, or tertiary expressions in C when given a single-use expression address. Requires outside analysis to provide the target expressions.

ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']

PLATFORMS = ['windows', 'linux', 'cgc']

STAGE: *OptimizationPassStage* = 10

NAME = 'Transform single-use expressions that were assigned to in different If-Else branches into ternary expressions'

DESCRIPTION = 'Transform specific expressions into If-Then-Else expressions, or tertiary expressions in C when\n given a single-use expression address. Requires outside analysis to provide the target expressions.'

__init__(**args*, *ite_exprs=None*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.ite_region_converter

class angr.analyses.decompiler.optimization_passes.ite_region_converter.**ITERegionConverter**

Bases: *OptimizationPass*

Transform regions of the form *if (c) {x = a} else {x = b}* into *x = c ? a : b*.

ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']

PLATFORMS = ['windows', 'linux', 'cgc']

STAGE: *OptimizationPassStage* = 6

NAME = 'Transform ITE-assignment regions into ternary expression assignments'

DESCRIPTION = 'Transform regions of the form `if (c) {x = a} else {x = b}` into `x = c ? a :  b`.'

__init__(**args*, *max_updates=10*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.known_pattern_outliner

class angr.analyses.decompiler.optimization_passes.known_pattern_outliner.**KnownPatternOutliner**

Bases: *OptimizationPass*

Automatically finds occurrences of KnownPatterns (inlined library idioms such as std::string::length() or std::vector<int>::size(); see angr.analyses.decompiler.known_patterns) in the AIL graph and outlines them into calls via the KnownPatternFinder analysis.

Runs right before variable recovery so that the synthesized calls' argument types flow into Typehoon within the same decompilation run.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 7

NAME = 'Outline known code patterns into calls'

DESCRIPTION = "Automatically finds occurrences of KnownPatterns (inlined library idioms\n such as std::string::length() or std::vector<int>::size(); see\n angr.analyses.decompiler.known_patterns) in the AIL graph and outlines them\n into calls via the KnownPatternFinder analysis.\n\n Runs right before variable recovery so that the synthesized calls' argument\n types flow into Typehoon within the same decompilation run."

MAX_ROUNDS = 8

__init__(*func*, *manager*, *known_patterns=None*, *recognize_known_patterns=True*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier

class angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.Case

Bases: object

Describes a case in a switch-case construct.

__init__(*original_node*, *node_type*, *variable_hash*, *expr*, *value*, *target*, *target_idx*, *next_addr*)

**Parameters**

- **node_type** (*str | None*)
- **value** (*int | str*)
- **target_idx** (*int | None*)

original_node

node_type

variable_hash

expr

value

target

target_idx

next_addr

class angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.StableVarExprHasher

Bases: *AILBlockViewer*

Obtain a stable hash of an AIL expression with respect to all variables and all operations applied on variables.

__init__(*expr*, *variable_map*)

**Parameters**

**expr** (Expression)

class angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier

Bases: *StructuringOptimizationPass*

This optimization recognizes and reverts switch cases that have been lowered and possibly split into multiple if-else statements. This optimization, discussed in the USENIX 2024 paper SAILR, aims to undo the compiler

optimization known as "Switch Lowering", present in both GCC and Clang. An in-depth discussion of this optimization can be found in the paper or in our documentation of the optimization: https://github.com/mahaloz/sailr-eval/issues/14#issue-2232616411

Note, this optimization does not occur in MSVC, which uses a different optimization strategy for switch cases. As a hack for now, we only run this deoptimization on Linux binaries.

**PLATFORMS = ['linux']**

**NAME = 'Convert lowered switch-cases (if-else) to switch-cases'**

**DESCRIPTION = 'Convert lowered switch-cases (if-else) to switch-cases. Only works when the Phoenix structuring algorithm is in use.'**

**__init__**(**args*, *min_distinct_cases=2*, ***kwargs*)

**static restore_graph**(*node*, *last_stmt*, *graph*, *full_graph*)

**Parameters**

- **last_stmt** (IncompleteSwitchCaseHeadStatement)
- **graph** (*DiGraph*)
- **full_graph** (*DiGraph*)

**static cases_issubset**(*cases_0*, *cases_1*)

Test if cases_0 is a subset of cases_1.

**Return type**
bool

**Parameters**

- **cases_0** (*list*[Case])
- **cases_1** (*list*[Case])

### angr.analyses.decompiler.optimization_passes.mips_gp_setting_simplifier

**class** angr.analyses.decompiler.optimization_passes.mips_gp_setting_simplifier.**MipsGpSettingSimplifier**

Bases: *OptimizationPass*

Removes $gp-setting statements at the beginning of MIPS functions.

**ARCHES = ['MIPS32', 'MIPS64']**

**PLATFORMS = ['linux']**

**STAGE: *OptimizationPassStage* = 6**

**NAME = 'Remove MIPS $gp-setting statements'**

**DESCRIPTION = 'Removes $gp-setting statements at the beginning of MIPS functions.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.mod_simplifier

**class** angr.analyses.decompiler.optimization_passes.mod_simplifier.**ModSimplifierAILEngine**

Bases: *SimplifierAILEngine*

**class** angr.analyses.decompiler.optimization_passes.mod_simplifier.**ModSimplifier**

Bases: *OptimizationPass*

Simplifies optimized forms of modulo computation back to "mod".

**ARCHES = ['X86', 'AMD64', 'ARMCortexM', 'ARMHF', 'ARMEL']**

**PLATFORMS = ['linux', 'windows']**

**STAGE: *OptimizationPassStage* = 6**

**NAME = 'Simplify optimized mod forms'**

**DESCRIPTION = 'Simplifies optimized forms of modulo computation back to "mod".'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.optimization_pass

**class** angr.analyses.decompiler.optimization_passes.optimization_pass.**BlockCache**

Bases: tuple

BlockCache(rd, prop)

**static __new__**(*_cls*, *rd*, *prop*)

Create new instance of BlockCache(rd, prop)

**prop**

Alias for field number 1

**rd**

Alias for field number 0

**exception**
angr.analyses.decompiler.optimization_passes.optimization_pass.**MultipleBlocksException**

Bases: Exception

An exception that is raised in _get_block() where multiple blocks satisfy the criteria but only one block was requested.

**class**
angr.analyses.decompiler.optimization_passes.optimization_pass.**OptimizationPassStage**

Bases: Enum

Enums about optimization pass stages.

Note that the region identification pass (RegionIdentifier) may modify existing AIL blocks *without updating the topology of the original AIL graph*. For example, loop successor refinement may modify create a new AIL block with an artificial address, and alter existing jump targets of jump statements and conditional jump statements to point to this new block. However, loop successor refinement does not update the topology of the original AIL graph, which means this new AIL block does not exist in the original AIL graph. As a result, until this behavior of RegionIdentifier changes in the future, DURING_REGION_IDENTIFICATION optimization passes should not modify existing jump targets.

AFTER_AIL_GRAPH_CREATION = 0

BEFORE_SSA_LEVEL0_TRANSFORMATION = 1

AFTER_SINGLE_BLOCK_SIMPLIFICATION = 2

BEFORE_SSA_LEVEL1_TRANSFORMATION = 3

AFTER_SSA_LEVEL1_TRANSFORMATION = 4

AFTER_MAKING_CALLSITES = 5

AFTER_GLOBAL_SIMPLIFICATION = 6

BEFORE_VARIABLE_RECOVERY = 7

AFTER_VARIABLE_RECOVERY = 8

BEFORE_REGION_IDENTIFICATION = 9

DURING_REGION_IDENTIFICATION = 10

AFTER_STRUCTURING = 11

class angr.analyses.decompiler.optimization_passes.optimization_pass.**BaseOptimizationPass**

Bases: object

The base class for any optimization pass.

ARCHES = []

PLATFORMS = []

STAGE: *OptimizationPassStage*

STRUCTURING: list[str] | None = None

NAME = 'N/A'

DESCRIPTION = 'N/A'

__init__(*func*, *manager*)

**Parameters**
**manager** (*Manager*)

property project: *Project*

property kb

analyze()

class angr.analyses.decompiler.optimization_passes.optimization_pass.**OptimizationPass**

Bases: *BaseOptimizationPass*

The base class for any function-level graph optimization pass.

__init__(*func*, *manager*, *, *graph*, *blocks_by_addr=None*, *blocks_by_addr_and_idx=None*, *kb=None*, *region_identifier=None*, *reaching_definitions=None*, *vvar_id_start=0*, *entry_node_addr=None*, *scratch=None*, *force_loop_single_exit=True*, *refine_loops_with_single_successor=False*, *expose_loop_head_backedges=False*, *fold_expressions=True*, *avoid_vvar_ids=None*, *arg_vvars=None*, *peephole_optimizations=None*, *stack_pointer_tracker=None*, *notes=None*, ***kwargs*)

**Parameters**

- **vvar_id_start** (*int*)
- **scratch** (*dict[str, Any] | None*)
- **force_loop_single_exit** (*bool*)
- **refine_loops_with_single_successor** (*bool*)
- **expose_loop_head_backedges** (*bool*)
- **fold_expressions** (*bool*)
- **avoid_vvar_ids** (*set[int] | None*)
- **arg_vvars** (*dict[int, tuple[VirtualVariable, SimVariable]] | None*)
- **notes** (*dict | None*)

**entry_node_addr:** tuple[int, int | None]

**out_graph:** DiGraph | None

**stack_items:** dict[int, *StackItem*]

**analyze**()

**property blocks_by_addr:** dict[int, set[Block]]

**property blocks_by_addr_and_idx:** dict[tuple[int, int | None], Block]

**bfs_nodes**(*depth=None*, *start_node=None*)

**Return type**
Generator[Block]

**Parameters**

- **depth** (*int | None*)
- **start_node** (*Block | None*)

**new_block_addr**()

Return a block address that does not conflict with any existing blocks.

**Return type**
int

**Returns**
The block address.

**class** angr.analyses.decompiler.optimization_passes.optimization_pass.**SequenceOptimizationPass**

Bases: *BaseOptimizationPass*

The base class for any sequence node optimization pass.

__init__(*func*, *manager*, *seq=None*, ***kwargs*)

class angr.analyses.decompiler.optimization_passes.optimization_pass.StructuringOptimizationPass

Bases: *OptimizationPass*

The base class for any optimization pass that requires structuring. Optimization passes that inherit from this class should directly depend on structuring artifacts, such as regions and gotos. Otherwise, they should use OptimizationPass. This is the heaviest (computation time) optimization pass class.

By default this type of optimization should work: - on any architecture - on any platform - during region identification (to have iterative structuring) - only with the SAILR structuring algorithm

ARCHES = None

PLATFORMS = None

STRUCTURING: list[str] | None = ['sailr']

STAGE: *OptimizationPassStage* = 10

__init__(*func*, *manager*, *require_structurable_graph=True*, *prevent_new_gotos=True*, *strictly_less_gotos=False*, *recover_structure_fails=True*, *must_improve_rel_quality=True*, *max_opt_iters=1*, *simplify_ail=True*, *require_gotos=True*, *readd_labels=False*, *edges_to_remove=None*, ***kwargs*)

**Parameters**

- **require_structurable_graph** (*bool*)
- **prevent_new_gotos** (*bool*)
- **strictly_less_gotos** (*bool*)
- **recover_structure_fails** (*bool*)
- **must_improve_rel_quality** (*bool*)
- **max_opt_iters** (*int*)
- **simplify_ail** (*bool*)
- **require_gotos** (*bool*)
- **readd_labels** (*bool*)
- **edges_to_remove** (*list[tuple[tuple[int, int | None], tuple[int, int | None]]] | None*)

analyze()

### angr.analyses.decompiler.optimization_passes.peephole_simplifier

class angr.analyses.decompiler.optimization_passes.peephole_simplifier.ExpressionSequenceWalker

Bases: *SequenceWalker*

Walks sequences with generic expression handling.

class angr.analyses.decompiler.optimization_passes.peephole_simplifier.
**PostStructuringPeepholeOptimizationPass**

Bases: *SequenceOptimizationPass*

Perform a post-structuring peephole optimization pass to simplify node statements and expressions.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 11**

**NAME = 'Post-Structuring Peephole Optimization'**

**DESCRIPTION = 'Perform a post-structuring peephole optimization pass to simplify node statements and expressions.'**

**__init__**(**args*, *peephole_optimizations=None*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.register_save_area_simplifier

class angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.
**RegisterSaveAreaSimplifier**

Bases: *OptimizationPass*

Optimizes away register spilling effects, including callee-saved registers.

This optimization runs between SSA-level0 and SSA-level1, which means registers are converted to vvars but stack accesses stay unchanged.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 2**

**NAME = 'Simplify register save areas'**

**DESCRIPTION = 'Optimizes away register spilling effects, including callee-saved registers.\n\n This optimization runs between SSA-level0 and SSA-level1, which means registers are converted to vvars but stack\n accesses stay unchanged.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.register_save_area_simplifier_adv

class angr.analyses.decompiler.optimization_passes.register_save_area_simplifier_adv.
**RegisterSaveAreaSimplifierAdvanced**

Bases: *OptimizationPass*

Optimizes away registers that are stored to or restored on the stack space.

This analysis is more complex than RegisterSaveAreaSimplifier because it handles: (1) Registers that are stored in the stack shadow space (sp+N) according to the Windows x64 calling convention. (2) Registers that are aliases of sp.

**ARCHES = None**

PLATFORMS = None

STAGE: *OptimizationPassStage* = 5

NAME = 'Simplify register save areas (advanced)'

DESCRIPTION = 'Optimizes away registers that are stored to or restored on the stack space.\n\n This analysis is more complex than RegisterSaveAreaSimplifier because it handles:\n (1) Registers that are stored in the stack shadow space (sp+N) according to the Windows x64 calling convention.\n (2) Registers that are aliases of sp.'

__init__(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier

class angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.**RetAddrSaveSimplifier**

Bases: *OptimizationPass*

Removes code in function prologues and epilogues for saving and restoring return address registers (ra, lr, etc.), generally seen in non-leaf functions.

ARCHES = ['MIPS32', 'MIPS64']

PLATFORMS = ['linux']

STAGE: *OptimizationPassStage* = 6

NAME = 'Simplify return address storage'

DESCRIPTION = 'Removes code in function prologues and epilogues for saving and restoring return address registers (ra, lr, etc.),\n generally seen in non-leaf functions.'

__init__(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.ret_deduplicator

class angr.analyses.decompiler.optimization_passes.ret_deduplicator.**ReturnDeduplicator**

Bases: *OptimizationPass*

Transforms: - if (cond) { … return x; } return x;

into: - if (cond) { … } return x;

TODO: its possible that this can be expanded to all rets that are equivalent. Testing needed.

ARCHES = ['X86', 'AMD64', 'ARMEL', 'ARMHF', 'ARMCortexM', 'MIPS32', 'MIPS64']

PLATFORMS = ['windows', 'linux', 'cgc']

STAGE: *OptimizationPassStage* = 10

NAME = 'Deduplicates return statements that may have been duplicated'

DESCRIPTION = 'Transforms:\n - if (cond) { ... return x; } return x;\n\n into:\n - if (cond) { ... } return x;\n\n TODO: its possible that this can be expanded to all rets that are equivalent. Testing needed.'

STRUCTURING: list[str] | None = ['sailr', 'dream']

__init__(*args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.return_duplicator_base

class angr.analyses.decompiler.optimization_passes.return_duplicator_base.FreshVirtualVariableRewriter

Bases: *AILBlockRewriter*

Helper class to rewrite virtual variables so that they will use fresh virtual variables.

__init__(*vvar_id_start*, *vvar_mapping*)

**Parameters**

- **vvar_id_start** (*int*)
- **vvar_mapping** (*dict[int, int]*)

class angr.analyses.decompiler.optimization_passes.return_duplicator_base.ReturnDuplicatorBase

Bases: object

The base class for implementing Return Duplication as described in the SAILR paper. This base class describes the general algorithm for duplicating return regions in a graph.

__init__(*func*, *manager*, *, *vvar_id_start*, *max_calls_in_regions=2*, *minimize_copies_for_regions=True*, *ri=None*, *scratch=None*, *max_func_blocks=1500*)

**Parameters**

- **vvar_id_start** (*int*)
- **max_calls_in_regions** (*int*)
- **minimize_copies_for_regions** (*bool*)
- **ri** (*RegionIdentifier | None*)
- **scratch** (*dict[str, Any] | None*)
- **max_func_blocks** (*int*)

next_node_idx()

**Return type**
int

static unwrap_conv(*expr*)

### angr.analyses.decompiler.optimization_passes.return_duplicator_high

class angr.analyses.decompiler.optimization_passes.return_duplicator_high.ReturnDuplicatorHigh

Bases: *OptimizationPass*, *ReturnDuplicatorBase*

This is a light-level goto-less version of the ReturnDuplicator optimization pass. It will only duplicate return-only blocks.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 6

NAME = 'Duplicate return-only blocks (high)'

DESCRIPTION = '\n This is a light-level goto-less version of the ReturnDuplicator optimization pass. It will only\n duplicate return-only blocks.\n '

STRUCTURING: list[str] | None = ['sailr', 'dream']

__init__(**args*, *vvar_id_start*, *max_calls_in_regions=2*, *minimize_copies_for_regions=True*, *scratch=None*, ***kwargs*)

**Parameters**

- **vvar_id_start** (*int*)
- **max_calls_in_regions** (*int*)
- **minimize_copies_for_regions** (*bool*)
- **scratch** (*dict[str, Any] | None*)

### angr.analyses.decompiler.optimization_passes.return_duplicator_low

class angr.analyses.decompiler.optimization_passes.return_duplicator_low.**ReturnDuplicatorLow**

Bases: *StructuringOptimizationPass*, *ReturnDuplicatorBase*

An optimization pass that reverts a subset of Irreducible Statement Condensing (ISC) optimizations, as described in the USENIX 2024 paper SAILR. This is the heavy/goto version of the ReturnDuplicator optimization pass.

Some compilers, including GCC, Clang, and MSVC, apply various optimizations to reduce the number of statements in code. These optimizations will take equivalent statements, or a subset of them, and replace them with a single copy that is jumped to by gotos – optimizing for space and sometimes speed.

This optimization pass will revert those gotos by re-duplicating the condensed blocks. Since Return statements are the most common, we use this optimization pass to revert only gotos to return statements. Additionally, we perform some additional readability fixups, like not re-duplicating returns to shared components.

**Parameters**

- **func** – The function to optimize.
- **node_idx_start** – The index to start at when creating new nodes. This is used by Clinic to ensure that node indices are unique across multiple passes.
- **max_opt_iters** (int) – The maximum number of optimization iterations to perform.
- **max_calls_in_regions** (int) – The maximum number of calls that can be in a region. This is used to prevent duplicating too much code.
- **prevent_new_gotos** (bool) – If True, this optimization pass will prevent new gotos from being created.
- **minimize_copies_for_regions** (bool) – If True, this optimization pass will minimize the number of copies by doing only a single copy for connected in_edges that form a region.

ARCHES = None

PLATFORMS = None

NAME = 'Duplicate returns connect with gotos (low)'

DESCRIPTION = 'An optimization pass that reverts a subset of Irreducible Statement Condensing (ISC) optimizations, as described\nin the USENIX 2024 paper SAILR. This is the heavy/goto version of the ReturnDuplicator optimization pass.\n\nSome compilers, including GCC, Clang, and MSVC, apply various optimizations to reduce the number of statements in\ncode. These optimizations will take equivalent statements, or a subset of them, and replace them with a single\ncopy that is jumped to by gotos -- optimizing for space and sometimes speed.\n\nThis optimization pass will revert those gotos by re-duplicating the condensed blocks. Since Return statements\nare the most common, we use this optimization pass to revert only gotos to return statements. Additionally, we\nperform some additional readability fixups, like not re-duplicating returns to shared components.'

__init__(**args*, *max_opt_iters=4*, *max_calls_in_regions=2*, *prevent_new_gotos=True*, *minimize_copies_for_regions=True*, *region_identifier=None*, *vvar_id_start=0*, *scratch=None*, *max_func_blocks=300*, ***kwargs*)

**Parameters**

- **max_opt_iters** (*int*)
- **max_calls_in_regions** (*int*)
- **prevent_new_gotos** (*bool*)
- **minimize_copies_for_regions** (*bool*)
- **vvar_id_start** (*int*)
- **scratch** (*dict[str, Any] | None*)
- **max_func_blocks** (*int*)

### angr.analyses.decompiler.optimization_passes.stack_canary_simplifier

class angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.**StackCanarySimplifier**

Bases: *OptimizationPass*

Removes stack canary checks from decompilation results.

ARCHES = ['X86', 'AMD64']

PLATFORMS = ['cgc', 'linux']

STAGE: *OptimizationPassStage* = 6

NAME = 'Simplify stack canaries'

DESCRIPTION = 'Removes stack canary checks from decompilation results.'

__init__(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.static_vvar_rewriter

class angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.**FixedBuffer**

Bases: object

A fixed-size buffer with known content.

__init__(*ident*, *size*, *content*)

**Parameters**

- **ident** (*str | None*)
- **size** (*int*)
- **content** (*bytes*)

class angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.**FixedBufferPtr**

Bases: object

A pointer to a fixed-size buffer.

__init__(*buffer_ident*, *offset=0*)

**Parameters**

- **buffer_ident** (*str*)
- **offset** (*int*)

class angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.**Offset**

Bases: object

Describes an offset value.

__init__(*value*, *bits*)

**Parameters**

- **value** (*int*)
- **bits** (*int*)

class angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.**VVarRewritingVisitor**

Bases: *AILBlockRewriter*

The visitor that rewrites vvars and their reads.

__init__(*static_buffers*, *static_vvars*, *kb*, *manager*)

**Parameters**

- **static_buffers** (*dict[str, FixedBuffer]*)
- **static_vvars** (*dict[int, FixedBufferPtr | Const]*)
- **manager** (*Manager*)

class angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.**VVarAliasVisitor**

Bases: *AILBlockViewer*

The visitor that discovers const assignments and aliases of existing static vvars.

__init__(*static_buffers*, *static_vvars*, *kb*, *manager*)

**Parameters**

- **static_buffers** (*dict[str, FixedBuffer]*)
- **static_vvars** (*dict[int, FixedBufferPtr | Const]*)
- **manager** (*Manager*)

class angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.**StaticVVarRewriter**

Bases: *OptimizationPass*

Rewrite user-specified vvars as static values or fix-sized buffers. Also rewrites reads from pointers derived off of such vvars.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Static virtual variable rewriter'**

**DESCRIPTION = 'Rewrite user-specified vvars as static values or fix-sized buffers.\nAlso rewrites reads from pointers derived off of such vvars.'**

**__init__**(**args*, *static_buffers=None*, *static_vvars=None*, ***kwargs*)

**Parameters**

- **static_buffers** (*dict[str, FixedBuffer] | None*)
- **static_vvars** (*dict[int, FixedBufferPtr | Const] | None*)

## angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator

class angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator.**SwitchDefaultCaseDuplicator**

Bases: *OptimizationPass*

For each switch-case construct (identified by jump tables), duplicate the default-case node when we detect situations where the default-case node is seemingly reused by edges outside the switch-case construct. This code reuse is usually caused by compiler code deduplication.

Ideally this pass should be implemented as an ISC optimization reversion.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 0**

**NAME = 'Duplicate default-case nodes to undo default-case node reuse caused by compiler code deduplication'**

**DESCRIPTION = 'For each switch-case construct (identified by jump tables), duplicate the default-case node when we detect\n situations where the default-case node is seemingly reused by edges outside the switch-case construct. This code\n reuse is usually caused by compiler code deduplication.\n\n Ideally this pass should be implemented as an ISC optimization reversion.'**

**__init__**(**args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.switch_reused_entry_rewriter

**class** angr.analyses.decompiler.optimization_passes.switch_reused_entry_rewriter.**SwitchReusedEntryRewriter**

Bases: *OptimizationPass*

For each switch-case construct (identified by jump tables), rewrite the entry into a goto block when we detect situations where an entry node is reused by edges in switch-case constructs that are not the current one. This code reuse is usually caused by compiler code deduplication.

The *max_entry_reuse_count* parameter controls the maximum number of times an entry can be reused in different jump tables in the same function; exceeding this threshold probably indicates that the jump table recovery is incorrect and thus creating jump tables with too many duplicate entries.

The *max_reused_entries* parameter controls the maximum number of reused entries that can be found in a function before aborting this optimization.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 0**

**NAME = 'Rewrite switch-case entry nodes with multiple predecessors into goto statements.'**

**DESCRIPTION = 'For each switch-case construct (identified by jump tables), rewrite the entry into a goto block when we detect\n situations where an entry node is reused by edges in switch-case constructs that are not the current one. This code\n reuse is usually caused by compiler code deduplication.\n\n The `max_entry_reuse_count` parameter controls the maximum number of times an entry can be reused in different jump\n tables in the same function; exceeding this threshold probably indicates that the jump table recovery is incorrect\n and thus creating jump tables with too many duplicate entries.\n\n The `max_reused_entries` parameter controls the maximum number of reused entries that can be found in a function\n before aborting this optimization.'**

**__init__**(**args*, *max_entry_reuse_count=10*, *max_reused_entries=20*, ***kwargs*)

**Parameters**

- **max_entry_reuse_count** (*int*)
- **max_reused_entries** (*int*)

### angr.analyses.decompiler.optimization_passes.tag_slicer

**class** angr.analyses.decompiler.optimization_passes.tag_slicer.**TagSlicer**

Bases: *OptimizationPass*

Removes unmarked statements from the graph.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 8**

**NAME = 'Remove unmarked statements from the graph.'**

DESCRIPTION = 'Removes unmarked statements from the graph.'

__init__(*args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier

class angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier.WinStackCanarySimplifier

Bases: *OptimizationPass*

Removes stack canary checks from decompilation results for Windows PE files.

we need to run this pass before performing any full-function simplification. Otherwise the effects of _security_cookie will be propagated.

ARCHES = ['X86', 'AMD64']

PLATFORMS = ['windows']

STAGE: *OptimizationPassStage* = 2

NAME = 'Simplify stack canaries in Windows PE files'

DESCRIPTION = 'Removes stack canary checks from decompilation results for Windows PE files.\n\n we need to run this pass before performing any full-function simplification. Otherwise the effects of\n _security_cookie will be propagated.'

__init__(*args*, ***kwargs*)

### angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier

class angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier

Bases: *OptimizationPass*

Simplifies __x86.get_pc_thunk calls.

ARCHES = ['X86']

PLATFORMS = ['linux']

STAGE: *OptimizationPassStage* = 1

NAME = 'Simplify getpc()'

DESCRIPTION = 'Simplifies __x86.get_pc_thunk calls.'

__init__(*args*, ***kwargs*)

### angr.analyses.decompiler.peephole_optimizations

#### Submodules

| |
|---|
| *a_div_const_add_a_mul_n_div_const* |
| *a_mul_const_div_shr_const* |
| *a_mul_const_sub_a* |

continues on next page

Table 19 – continued from previous page

| |
|---|
| *a_shl_const_sub_a* |
| *a_sub_a_div* |
| *a_sub_a_shr_const_shr_const* |
| *a_sub_a_sub_n* |
| *arm_cmpf* |
| *base* |
| *basepointeroffset_add_n* |
| *basepointeroffset_and_mask* |
| *bitwise_inserts* |
| *bitwise_or_to_logical_or* |
| *bool_expr_xor_1* |
| *bswap* |
| *bt_rflags_merge* |
| *cas_intrinsics* |
| *cmp_masked_shift* |
| *cmp_sub_const* |
| *cmpord_rewriter* |
| *coalesce_adjacent_shrs* |
| *coalesce_same_cascading_ifs* |
| *concat_simplifier* |
| *constant_derefs* |
| *conv_a_sub0_shr_and* |
| *conv_shl_shr* |
| *eager_eval* |
| *evaluate_const_conversions* |
| *extended_byte_and_mask* |
| *inlined_strcpy* |
| *inlined_strcpy_consolidation* |
| *invert_negated_logical_conjuction_disjunctio* |
| *known_pattern_call_info* |
| *modulo_simplifier* |
| *one_sub_bool* |
| *optimized_div_simplifier* |
| *remove_cascading_conversions* |
| *remove_const_insert* |
| *remove_cxx_destructor_calls* |
| *remove_empty_if_body* |
| *remove_noop_conversions* |
| *remove_redundant_bitmasks* |
| *remove_redundant_conversions* |
| *remove_redundant_derefs* |
| *remove_redundant_insert* |
| *remove_redundant_ite_branch* |
| *remove_redundant_ite_comparisons* |
| *remove_redundant_nots* |
| *remove_redundant_reinterprets* |
| *remove_redundant_shifts* |
| *remove_redundant_shifts_around_comparators* |
| *rewrite_bit_extractions* |
| *rewrite_conv_mul* |
| *rewrite_cxx_operator_calls* |
| *rewrite_mips_gp_loads* |

continues on next page

Table 19 – continued from previous page

| |
|---|
| *rol_ror* |
| *sar_to_signed_div* |
| *shl_to_mul* |
| *simplify_pc_relative_loads* |
| *single_bit_cond_to_boolexpr* |
| *single_bit_xor* |
| *tidy_stack_addr* |
| *utils* |

### angr.analyses.decompiler.peephole_optimizations.a_div_const_add_a_mul_n_div_const

**class** angr.analyses.decompiler.peephole_optimizations.a_div_const_add_a_mul_n_div_const.**ADivConstAddAMulNDivConst**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'a / N0 + (a * N1) / N0 => a * (N1 + 1) / N0'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.a_mul_const_div_shr_const

**class** angr.analyses.decompiler.peephole_optimizations.a_mul_const_div_shr_const.**AMulConstDivShrConst**

Bases: *PeepholeOptimizationExprBase*

**NAME = '(A * N0 / N1) >> N2 => (A * (N0 / 2 ** N2) / N1)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.a_mul_const_sub_a

**class** angr.analyses.decompiler.peephole_optimizations.a_mul_const_sub_a.**AMulConstSubA**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'a * N - a => a * (N - 1)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.a_shl_const_sub_a

class angr.analyses.decompiler.peephole_optimizations.a_shl_const_sub_a.**AShlConstSubA**

Bases: *PeepholeOptimizationExprBase*

**NAME** = '(a << N) - a => (a * (2 ** N - 1))'

**expr_classes** = (<class 'angr.ailment.expression.BinaryOp'>,)

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.a_sub_a_div

class angr.analyses.decompiler.peephole_optimizations.a_sub_a_div.**ASubADiv**

Bases: *PeepholeOptimizationExprBase*

**NAME** = 'a - a / N => a * (N - 1) / N'

**expr_classes** = (<class 'angr.ailment.expression.BinaryOp'>,)

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.a_sub_a_shr_const_shr_const

class angr.analyses.decompiler.peephole_optimizations.a_sub_a_shr_const_shr_const.**ASubAShrConstShrConst**

Bases: *PeepholeOptimizationExprBase*

Convert *cdq; sub eax, edx; sar eax, 1* to *eax /= 2*.

**NAME** = '(a - (a >> 31)) >> N => a / 2 ** N (signed)'

**expr_classes** = (<class 'angr.ailment.expression.BinaryOp'>,)

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.a_sub_a_sub_n

class angr.analyses.decompiler.peephole_optimizations.a_sub_a_sub_n.**ASubASubN**

Bases: *PeepholeOptimizationExprBase*

**NAME** = 'expr - (expr - N) => N'

**expr_classes** = (<class 'angr.ailment.expression.BinaryOp'>,)

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.arm_cmpf

**class** angr.analyses.decompiler.peephole_optimizations.arm_cmpf.**ARMCmpF**

Bases: *PeepholeOptimizationExprBase*

Optimizes floating-point comparison expressions.

**NAME = 'Simplifying CmpF on ARM'**

**expr_classes = (<class 'angr.ailment.expression.Convert'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Convert)

### angr.analyses.decompiler.peephole_optimizations.base

**class** angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationStmtBase**

Bases: object

The base class for all peephole optimizations that are applied on AIL statements.

fixpoint_reached is an output parameter; it tells the caller whether this optimizer should run on this statement again (e.g., when statements prior to the current statement are optimized and changed). The caller sets fixpoint_reached to True before every *optimize()* call; the optimizer optionally sets it to False if it wants to be invoked again on the same statement.

So, fixpoint_reached should be set to (or kept) True if (a) the optimizer has optimized the statement and does not expect to optimize it ever again, or (b) the optimizer cannot ever optimize the statement.

**NAME = 'Peephole Optimization - Statement'**

**DESCRIPTION = 'Peephole Optimization - Statement'**

**stmt_classes = None**

**__init__**(*project*, *kb*, *ail_manager*, *func_addr=None*, *preserve_vvar_ids=None*, *type_hints=None*)

**Parameters**

- **project** (*Project | None*)
- **kb** (*KnowledgeBase | None*)
- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)

**project: *Project* | None**

**kb: *KnowledgeBase* | None**

**manager**

**func_addr: int | None**

**preserve_vvar_ids:** *set[int]*

**type_hints:** *list[tuple[VirtualVariable | MemoryLocation, str]]*

**fixpoint_reached:** *bool*

**optimize**(*stmt*, *stmt_idx*, *block*, ***kwargs*)

**Parameters**

- **stmt_idx** (*int*)
- **block** (*Block*)

**class** angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationMultiStmtBase**

Bases: object

The base class for all peephole optimizations that are applied on multiple AIL statements at once.

fixpoint_reached exists for uniformity but is unused. Multi-statement optimizers always run regardless of whether the statements have reached fixed points or not.

**NAME = 'Peephole Optimization - Multi-statement'**

**DESCRIPTION = 'Peephole Optimization - Multi-statement'**

**stmt_classes = None**

**__init__**(*project*, *kb*, *ail_manager*, *func_addr=None*, *preserve_vvar_ids=None*, *type_hints=None*)

**Parameters**

- **project** (*Project | None*)
- **kb** (*KnowledgeBase | None*)
- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)

**project:** *Project | None*

**kb:** *KnowledgeBase | None*

**manager**

**func_addr:** *int | None*

**preserve_vvar_ids:** *set[int]*

**type_hints:** *list[tuple[VirtualVariable | MemoryLocation, str]]*

**fixpoint_reached:** *bool*

**optimize**(*stmts*, *stmt_idx=None*, *block=None*, ***kwargs*)

**Parameters**

- **stmts** (*list[Statement]*)
- **stmt_idx** (*int | None*)

class angr.analyses.decompiler.peephole_optimizations.base.**PeepholeOptimizationExprBase**

Bases: object

The base class for all peephole optimizations that are applied on AIL expressions. Please refer to *PeepholeOptimizationStmtBase* for the `fixpoint_reached` contract.

**NAME = 'Peephole Optimization - Expression'**

**DESCRIPTION = 'Peephole Optimization - Expression'**

**expr_classes = None**

**__init__**(*project*, *kb*, *ail_manager*, *func_addr=None*, *preserve_vvar_ids=None*, *type_hints=None*)

**Parameters**

- **project** (*Project | None*)
- **kb** (*KnowledgeBase | None*)
- **ail_manager** (*Manager*)
- **func_addr** (*int | None*)
- **preserve_vvar_ids** (*set[int] | None*)
- **type_hints** (*list[tuple[VirtualVariable | MemoryLocation, str]] | None*)

**project: *Project* | None**

**kb: *KnowledgeBase* | None**

**manager**

**func_addr: int | None**

**preserve_vvar_ids: set[int]**

**type_hints: list[tuple[*VirtualVariable* | *MemoryLocation*, str]]**

**fixpoint_reached: bool**

**optimize**(*expr*, ***, *stmt_idx=None*, *block=None*, ***kwargs*)

**Return type**
*Expression* | None

**Parameters**
**stmt_idx** (*int | None*)

**static find_definition**(*ail_expr*, *stmt_idx*, *block*)

**Return type**
None

**Parameters**

- **ail_expr** (Expression)
- **stmt_idx** (*int*)
- **block** (*Block*)

**static is_bool_expr**(*ail_expr*)

### angr.analyses.decompiler.peephole_optimizations.basepointeroffset_add_n

**class** angr.analyses.decompiler.peephole_optimizations.basepointeroffset_add_n.**BasePointerOffsetAddN**

Bases: *PeepholeOptimizationExprBase*

**NAME = '(Ptr - M) + N => Ptr - (M - N)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.basepointeroffset_and_mask

**class** angr.analyses.decompiler.peephole_optimizations.basepointeroffset_and_mask.**BasePointerOffsetAndMask**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'Ptr & mask => Ptr'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.bitwise_inserts

**class** angr.analyses.decompiler.peephole_optimizations.bitwise_inserts.**SimplifyBitwiseInserts**

Bases: *PeepholeOptimizationExprBase*

simplifies _INSERT(A, c, EXTRACT(A, ...) | B) => A | B

**NAME = 'Simplify bitwise inserts'**

**expr_classes = (<class 'angr.ailment.expression.Insert'>,)**

**optimize**(*expr*, ***kwargs*)

### angr.analyses.decompiler.peephole_optimizations.bitwise_or_to_logical_or

**class** angr.analyses.decompiler.peephole_optimizations.bitwise_or_to_logical_or.**BitwiseOrToLogicalOr**

Bases: *PeepholeOptimizationExprBase*

Perform the following two simplifications:

1. (a | b) == 0 ==> (a == 0) && (b == 0)
2. (a | b) != 0 ==> (a != 0) || (b != 0)

**NAME = '(a | b) == 0 => (a == 0) && (b == 0) ; (a | b) != 0 => (a != 0) || (b != 0)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.bool_expr_xor_1

**class** angr.analyses.decompiler.peephole_optimizations.bool_expr_xor_1.**BoolExprXor1**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'bool_expr ^ 1 => !bool_expr (a)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.bswap

**class** angr.analyses.decompiler.peephole_optimizations.bswap.**Bswap**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'Simplifying bswap_16(), bswap_32(), and bswap_64()'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>, <class 'angr.ailment.expression.Convert'>)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.bt_rflags_merge

**class** angr.analyses.decompiler.peephole_optimizations.bt_rflags_merge.**BTRflagsMergeBitExtraction**

Bases: *PeepholeOptimizationExprBase*

Fold CF extractions from the flags merge that the libVEX 3.27+ amd64 BT/BTS/BTR/BTC lifting emits.

The new lifting preserves O/S/A/P/Z across BT-family instructions by writing `(amd64g_calculate_rflags_all(...) & 0x8D4) | bt_bit` into the flags thunk. A downstream condition that tests the tested bit (CF, bit 0) extracts it as `(merge & 0x8D4 | bt_bit) & 1`; since 0x8D4 has bit 0 cleared, this is just `bt_bit`, and folding it lets the (otherwise unreducible) rflags_all ccall become dead.

**NAME = '(x & 0x8d4 | bit) & 1 => bit'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.cas_intrinsics

angr.analyses.decompiler.peephole_optimizations.cas_intrinsics.**cas_intrinsic_name**(*mnemonic*, *os_name*)

Resolve the intrinsic name for a lock-prefixed instruction on the given OS, falling back to Linux naming.

**Return type**
str

**Parameters**

- **mnemonic** (*str*)
- **os_name** (*str | None*)

**class** angr.analyses.decompiler.peephole_optimizations.cas_intrinsics.**CASIntrinsics**

Bases: *PeepholeOptimizationMultiStmtBase*

Rewrite lock-prefixed instructions (or rather, their VEX/AIL forms) into intrinsic calls.

Case 1:

```
             mov eax, r12d
0x140014b57: xchg eax, [0x14000365f8]

LABEL_0x140014b57:
CAS(0x1400365f8<64>, Conv(64->32, vvar_365{reg 112}), Load(addr=0x1400365f8<64>,␣
↪size=4, endness=Iend_LE),
    vvar_27756)
if (CasCmpNE(vvar_27756, g_1400365f8))
    goto LABEL_0x140014b57;

=> vvar_27756 = _InterlockedExchange(0x1400365f8, vvar_365{reg 112})
```

Case 2:

```
lock cmpxchg cs:g_WarbirdSecureFunctionsLock, r14d

CAS(0x1400365f8<64>, 0x1<32>, 0x0<32>, vvar_27751)

=> var_27751 = _InterlockedCompareExchange(0x1400365f8, 0x1<32>, 0x0<32>)
```

**NAME = 'Rewrite compare-and-swap instructions into intrinsics.'**

**stmt_classes = ((<class 'angr.ailment.statement.CAS'>, <class 'angr.ailment.statement.ConditionalJump'>), (<class 'angr.ailment.statement.CAS'>, <class 'angr.ailment.statement.Statement'>))**

**optimize**(*stmts*, *stmt_idx=None*, *block=None*, ***kwargs*)

**Parameters**

- **stmts** (*list[Statement]*)
- **stmt_idx** (*int | None*)

### angr.analyses.decompiler.peephole_optimizations.cmp_masked_shift

**class** angr.analyses.decompiler.peephole_optimizations.cmp_masked_shift.**CmpMaskedShift**

Bases: *PeepholeOptimizationExprBase*

Rewrite an equality comparison against a right-shifted value back into a masked comparison, which reads better for flag/discriminant checks.

(x >> n) == c ==> (x & mask) == (c << n) Convert(N->M, x >> n) == c ==> (x & mask) == (c << n)

where `mask` selects the compared bits `[n, n+width-1]` of `x` and `width` is the truncated width `M` (or `N - n` when there is no Convert). This is the inverse of a simplification that can turn `(x & high_mask) == c` (a mask that clears the low bits) into the shift/extract form: the masked compare is the clearer form in decompiled output, so restore it.

As a special case, when the compared bits are *all* bits above `n` and the constant is 0, the comparison is an unsigned bound check (libVEX 3.27+ spechelpers fold `x <=u 2**n-1` into this form), so emit the bound check instead:

(x >> n) == 0 ==> x <=u 2**n - 1 (x >> n) != 0 ==> x >u 2**n - 1

Only logical right shifts and unsigned truncations are handled, and only `n > 0` (a low-mask compare, `n == 0`, already reads as a cast).

**NAME = '(x >> n) == c => (x & mask) == (c << n)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.cmp_sub_const

**class** angr.analyses.decompiler.peephole_optimizations.cmp_sub_const.**CmpSubConst**

Bases: *PeepholeOptimizationExprBase*

Canonicalize equality/inequality comparisons against a constant where one side is a constant add/subtract, by folding the inner constant into the compared constant. This undoes the compiler's strength-reduced sub/dec-cascade lowering of switch-on-value statements, turning relative constants back into absolute ones:

```
(A - C1) == C2      ==>  A == (C1 + C2)
(A - C1) != C2      ==>  A != (C1 + C2)
(C1 - A) == C2      ==>  A == (C1 - C2)
(A + C1) == C2      ==>  A == (C2 - C1)
```

Note that this rule is only applied to CmpEQ/CmpNE. Over the modular integers Z/2^n, `A - C1 == C2` iff `A == C1 + C2` and `A + C1 == C2` iff `A == C2 - C1` hold unconditionally, independent of signedness and regardless of any intermediate overflow. This rule is NOT applied to ordered comparisons (e.g., CmpLT), for which folding across a subtract is unsound because of wraparound.

**NAME = '(A - C1) cmp C2 => A cmp (C1 + C2)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.cmpord_rewriter

**class** angr.analyses.decompiler.peephole_optimizations.cmpord_rewriter.**CmpORDRewriter**

Bases: *PeepholeOptimizationStmtBase*

Rewrites CmpORD expressions (PowerPC and VEX-specific) into common comparison operations.

**NAME = 'CmpORD rewriter'**

**stmt_classes = (<class 'angr.ailment.statement.ConditionalJump'>,)**

**optimize**(*stmt*, *stmt_idx*, *block*, ***kwargs*)

**Parameters**

- **stmt** (ConditionalJump)
- **stmt_idx** (*int*)
- **block** (*Block*)

### angr.analyses.decompiler.peephole_optimizations.coalesce_adjacent_shrs

**class** angr.analyses.decompiler.peephole_optimizations.coalesce_adjacent_shrs.**CoalesceAdjacentShiftRights**

Bases: *PeepholeOptimizationExprBase*

Coalesce adjacent SHR/SAR operations if possible.

**NAME = 'Coalesce adjacent shr/sars'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**

**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.coalesce_same_cascading_ifs

**class** angr.analyses.decompiler.peephole_optimizations.coalesce_same_cascading_ifs.**CoalesceSameCascadingIfs**

Bases: *PeepholeOptimizationStmtBase*

**NAME = 'Coalescing cascading If constructs'**

**stmt_classes = (<class 'angr.ailment.statement.ConditionalJump'>,)**

**optimize**(*stmt*, *stmt_idx*, *block*, ***kwargs*)

**Parameters**

- **stmt** (ConditionalJump)
- **stmt_idx** (*int*)
- **block** (*Block*)

### angr.analyses.decompiler.peephole_optimizations.concat_simplifier

class angr.analyses.decompiler.peephole_optimizations.concat_simplifier.**ConcatSimplifier**

Bases: *PeepholeOptimizationExprBase*

Simplify Concat expressions: - (a CONCAT b) >> bits(b) => a (high-part extraction) - (a CONCAT b) & mask => b (low-part extraction, where mask = (1 << bits(b)) - 1) - (a >> (bits-1)) CONCAT a => Convert(a, signed, 2*bits) (sign-extension) - 0 CONCAT a => Convert(a, unsigned, 2*bits) (zero-extension) - Convert(a CONCAT b, to_bits=bits(b)) => b (truncate to low part)

**NAME = 'Simplify Concat expressions'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>, <class 'angr.ailment.expression.Convert'>)**

**optimize**(*expr*, *stmt_idx=None*, *block=None*, ***kwargs*)

**Parameters**

- **expr** (BinaryOp | Convert)
- **stmt_idx** (*int* | *None*)

### angr.analyses.decompiler.peephole_optimizations.constant_derefs

class angr.analyses.decompiler.peephole_optimizations.constant_derefs.**ConstantDereferences**

Bases: *PeepholeOptimizationExprBase*

Dereferences constant memory loads from read-only memory regions.

**NAME = 'Dereference constant references'**

**expr_classes = (<class 'angr.ailment.expression.Load'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Load)

### angr.analyses.decompiler.peephole_optimizations.conv_a_sub0_shr_and

class angr.analyses.decompiler.peephole_optimizations.conv_a_sub0_shr_and.**ConvASub0ShrAnd**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'Conv(M->1, (expr >> N) & 1) => expr < 0'**

**expr_classes = (<class 'angr.ailment.expression.Convert'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Convert)

### angr.analyses.decompiler.peephole_optimizations.conv_shl_shr

class angr.analyses.decompiler.peephole_optimizations.conv_shl_shr.**ConvShlShr**

Bases: *PeepholeOptimizationExprBase*

NAME = '(expr << P) >> Q => (expr & mask) >> R'

expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)

optimize(*expr*, ***kwargs*)

Parameters
expr (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.eager_eval

class angr.analyses.decompiler.peephole_optimizations.eager_eval.EagerEvaluation

Bases: *PeepholeOptimizationExprBase*

Eagerly evaluates certain types of expressions.

NAME = 'Eager expression evaluation'

expr_classes = (<class 'angr.ailment.expression.BinaryOp'>, <class 'angr.ailment.expression.UnaryOp'>, <class 'angr.ailment.expression.Convert'>, <class 'angr.ailment.expression.ITE'>)

optimize(*expr*, ***kwargs*)

### angr.analyses.decompiler.peephole_optimizations.evaluate_const_conversions

class angr.analyses.decompiler.peephole_optimizations.evaluate_const_conversions.EvaluateConstConversions

Bases: *PeepholeOptimizationExprBase*

If we see a conversion over a constant, simply evaluate it

DESCRIPTION = "Conv(*, C) => C'"

expr_classes = (<class 'angr.ailment.expression.Convert'>, <class 'angr.ailment.expression.Extract'>)

optimize(*expr*, ***, *stmt_idx=None*, *block=None*, ***kwargs*)

Parameters
stmt_idx (*int | None*)

fixpoint_reached: bool

func_addr: int | None

kb: *KnowledgeBase* | None

manager

preserve_vvar_ids: set[int]

project: *Project* | None

type_hints: list[tuple[*VirtualVariable* | *MemoryLocation*, str]]

### angr.analyses.decompiler.peephole_optimizations.extended_byte_and_mask

**class** angr.analyses.decompiler.peephole_optimizations.extended_byte_and_mask.
**ExtendedByteAndMask**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'extended byte & 0xff..ff => extended byte'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.inlined_strcpy

### angr.analyses.decompiler.peephole_optimizations.inlined_strcpy_consolidation

### angr.analyses.decompiler.peephole_optimizations.invert_negated_logical_conjuction_disjunction

**class** angr.analyses.decompiler.peephole_optimizations.
invert_negated_logical_conjuction_disjunction.
**InvertNegatedLogicalConjunctionsAndDisjunctions**

Bases: *PeepholeOptimizationExprBase*

Push negations into subexpressions of logical conjunctions and logical disjunctions.

**NAME = '!(A && B) => A || B; !(A || B) => A && B'**

**expr_classes = (<class 'angr.ailment.expression.UnaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (UnaryOp)

### angr.analyses.decompiler.peephole_optimizations.known_pattern_call_info

**class** angr.analyses.decompiler.peephole_optimizations.known_pattern_call_info.
**KnownPatternCallInfo**

Bases: *PeepholeOptimizationExprBase*

Attach call-site prototypes (and optional vvar type hints) to calls that were synthesized from KnownPatterns (see angr.analyses.decompiler.known_patterns).

Known-pattern calls are identified by their string target, which is stable across Clinic runs, while the VariableMap holding prototypes is scoped to a single Clinic run; running as a peephole re-applies the prototype in every run so that variable recovery emits per-argument type constraints for Typehoon (e.g. `arg <: std::string *` for std::string::length).

**NAME = 'Attach prototypes and type hints to known-pattern calls'**

**expr_classes = (<class 'angr.ailment.expression.Call'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Call)

### angr.analyses.decompiler.peephole_optimizations.modulo_simplifier

**class** angr.analyses.decompiler.peephole_optimizations.modulo_simplifier.**ModuloSimplifier**

Bases: *PeepholeOptimizationExprBase*

Simplify division and multiplication expressions that can be reduced to a modulo operation.

**NAME = 'a - (a / N) * N => a % N'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, *stmt_idx=None*, *block=None*, ***kwargs*)

**Parameters**

- **expr** (BinaryOp)
- **stmt_idx** (*int | None*)

### angr.analyses.decompiler.peephole_optimizations.one_sub_bool

**class** angr.analyses.decompiler.peephole_optimizations.one_sub_bool.**OneSubBool**

Bases: *PeepholeOptimizationExprBase*

**NAME = '1 - bool_expr => !bool_expr'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**

**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.optimized_div_simplifier

**class** angr.analyses.decompiler.peephole_optimizations.optimized_div_simplifier.**OptimizedDivisionSimplifier**

Bases: *PeepholeOptimizationExprBase*

Convert expressions with right shifts into expressions with divisions.

**NAME = 'Simplify optimized division expressions, e.g., (N * a) >> M => a / N1'**

**expr_classes = (<class 'angr.ailment.expression.Convert'>, <class 'angr.ailment.expression.BinaryOp'>)**

**optimize**(*expr*, *stmt_idx=None*, *block=None*, ***kwargs*)

**Parameters**

- **expr** (Convert | BinaryOp)
- **stmt_idx** (*int | None*)

**optimize_binaryop**(*original_expr*)

**Parameters**

**original_expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.remove_cascading_conversions

**class** angr.analyses.decompiler.peephole_optimizations.remove_cascading_conversions.**RemoveCascadingConversions**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'Simplify adjacent size conversions'**

**expr_classes = (<class 'angr.ailment.expression.Convert'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Convert)

### angr.analyses.decompiler.peephole_optimizations.remove_const_insert

**class** angr.analyses.decompiler.peephole_optimizations.remove_const_insert.**RemoveConstInsert**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'Insert(c0, c1, v) ==> (c0 & mask) | (v << c1)'**

**expr_classes = (<class 'angr.ailment.expression.Insert'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Insert)

### angr.analyses.decompiler.peephole_optimizations.remove_cxx_destructor_calls

**class** angr.analyses.decompiler.peephole_optimizations.remove_cxx_destructor_calls.**RemoveCxxDestructorCalls**

Bases: *PeepholeOptimizationStmtBase*

Rewrite C++ operator function calls into operations.

**NAME = 'Remove C++ destructor function calls'**

**stmt_classes = (<class 'angr.ailment.statement.SideEffectStatement'>,)**

**optimize**(*stmt*, ***kwargs*)

**Return type**
tuple | None

**Parameters**
**stmt** (SideEffectStatement)

### angr.analyses.decompiler.peephole_optimizations.remove_empty_if_body

**class** angr.analyses.decompiler.peephole_optimizations.remove_empty_if_body.**RemoveEmptyIfBody**

Bases: *PeepholeOptimizationStmtBase*

**NAME = 'Remove empty If bodies'**

stmt_classes = (<class 'angr.ailment.statement.ConditionalJump'>,)

optimize(*stmt*, *stmt_idx*, *block*, ***kwargs*)

**Parameters**

- **stmt** (ConditionalJump)
- **stmt_idx** (*int*)
- **block** (*Block*)

### angr.analyses.decompiler.peephole_optimizations.remove_noop_conversions

class angr.analyses.decompiler.peephole_optimizations.remove_noop_conversions.**RemoveNoopConversions**

Bases: *PeepholeOptimizationExprBase*

NAME = 'Remove no-op conversions'

expr_classes = (<class 'angr.ailment.expression.Convert'>, <class 'angr.ailment.expression.Extract'>)

optimize(*expr*, ***kwargs*)

**Parameters**
**expr** (Convert | Extract)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks

class angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.**RemoveRedundantBitmasks**

Bases: *PeepholeOptimizationExprBase*

Remove redundant bitmasking operations.

NAME = 'Remove redundant bitmasks'

expr_classes = (<class 'angr.ailment.expression.BinaryOp'>, <class 'angr.ailment.expression.Convert'>, <class 'angr.ailment.expression.Extract'>, <class 'angr.ailment.expression.Insert'>)

optimize(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp | Convert | Extract | Insert)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_conversions

class angr.analyses.decompiler.peephole_optimizations.remove_redundant_conversions.**RemoveRedundantConversions**

Bases: *PeepholeOptimizationExprBase*

NAME = 'Remove or rewrite redundant conversions around binary operators'

expr_classes = (<class 'angr.ailment.expression.BinaryOp'>, <class 'angr.ailment.expression.Convert'>)

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp | Convert)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_derefs

**class** angr.analyses.decompiler.peephole_optimizations.remove_redundant_derefs.**RemoveRedundantDerefs**

Bases: *PeepholeOptimizationExprBase*

Remove redundant dereferences (e.g. *(&v)).

**NAME = 'Remove redundant dereferences'**

**expr_classes = (<class 'angr.ailment.expression.Load'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Load)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_insert

**class** angr.analyses.decompiler.peephole_optimizations.remove_redundant_insert.**RemoveRedundantInsert**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'Remove redundant INSERT operators'**

**expr_classes = (<class 'angr.ailment.expression.Insert'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Insert)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_branch

**class** angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_branch.**RemoveRedundantITEBranches**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'Remove redundant ITE branches'**

**expr_classes = (<class 'angr.ailment.expression.ITE'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (ITE)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_comparisons

class angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_comparisons.RemoveRedundantITEComparisons

Bases: PeepholeOptimizationExprBase

Remove redundant ITE comparisons.

NAME = 'Remove redundant ITE comparisons'

expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)

optimize(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_nots

class angr.analyses.decompiler.peephole_optimizations.remove_redundant_nots.RemoveRedundantNots

Bases: PeepholeOptimizationExprBase

NAME = 'Remove redundant Nots'

expr_classes = (<class 'angr.ailment.expression.UnaryOp'>,)

optimize(*expr*, ***kwargs*)

**Parameters**
**expr** (UnaryOp)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_reinterprets

class angr.analyses.decompiler.peephole_optimizations.remove_redundant_reinterprets.RemoveRedundantReinterprets

Bases: PeepholeOptimizationExprBase

Simplify nested and constant Reinterpret() expressions.

NAME = 'Simplifying nested and constant Reinterprets'

expr_classes = (<class 'angr.ailment.expression.Reinterpret'>,)

optimize(*expr*, ***kwargs*)

**Parameters**
**expr** (Reinterpret)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts

class angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts.RemoveRedundantShifts

Bases: PeepholeOptimizationExprBase

Remove redundant bitshift operations.

NAME = 'Remove redundant bitshifts'

expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts_around_comparators

**class** angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts_around_comparators.**RemoveRedundantShiftsAroundComparators**

Bases: *PeepholeOptimizationExprBase*

Remove redundant bitshifts for both operands around a comparison operator.

More cases can be added in the future as we encounter them.

**NAME = 'Remove redundant bitshifts for operands around a comparator'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, *stmt_idx=None*, *block=None*, ***kwargs*)

**Parameters**

- **expr** (BinaryOp)
- **stmt_idx** (*int | None*)

### angr.analyses.decompiler.peephole_optimizations.rewrite_bit_extractions

**class** angr.analyses.decompiler.peephole_optimizations.rewrite_bit_extractions.**RewriteBitExtractions**

Bases: *PeepholeOptimizationExprBase*

Rewrite bit extraction expressions

**NAME = 'Bit-extraction Rewriter'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.rewrite_conv_mul

**class** angr.analyses.decompiler.peephole_optimizations.rewrite_conv_mul.**RewriteConvMul**

Bases: *PeepholeOptimizationExprBase*

Rewrites multiplication to be inside conversion.

**NAME = 'Rewrite Conv Mul'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.rewrite_cxx_operator_calls

**class** angr.analyses.decompiler.peephole_optimizations.rewrite_cxx_operator_calls.**RewriteCxxOperatorCalls**

Bases: *PeepholeOptimizationStmtBase*

Rewrite C++ operator function calls into operations.

**NAME = 'Rewrite C++ operator function calls into operations'**

**stmt_classes = (<class 'angr.ailment.statement.SideEffectStatement'>, <class 'angr.ailment.statement.WeakAssignment'>)**

**optimize**(*stmt*, *block=None*, ***kwargs*)

**Parameters**
**stmt** (SideEffectStatement | WeakAssignment)

### angr.analyses.decompiler.peephole_optimizations.rewrite_mips_gp_loads

**class** angr.analyses.decompiler.peephole_optimizations.rewrite_mips_gp_loads.**RewriteMipsGpLoads**

Bases: *PeepholeOptimizationExprBase*

Rewrite $gp-based loads to their actual values on MIPS.

**NAME = 'MIPS GP-based Loads Rewriter'**

**expr_classes = (<class 'angr.ailment.expression.Load'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Load)

### angr.analyses.decompiler.peephole_optimizations.rol_ror

**class** angr.analyses.decompiler.peephole_optimizations.rol_ror.**RolRorRewriter**

Bases: *PeepholeOptimizationStmtBase*

Rewrites consecutive statements into ROL (rotate shift left) or ROR (rotate shift right) statements.

**NAME = 'ROL/ROR rewriter'**

**stmt_classes = (<class 'angr.ailment.statement.Assignment'>,)**

**optimize**(*stmt*, *stmt_idx*, *block*, ***kwargs*)

**Parameters**

- **stmt** (Assignment)
- **stmt_idx** (*int*)
- **block** (*Block*)

### angr.analyses.decompiler.peephole_optimizations.sar_to_signed_div

class angr.analyses.decompiler.peephole_optimizations.sar_to_signed_div.**SarToSignedDiv**

Bases: *PeepholeOptimizationExprBase*

Simplify signed divisions that are optimized into bit shifts during compilation.

**NAME = '(signed(expr)? expr + A ** 2 - 1:  expr) >>s A => expr /s 2 ** A'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, *stmt_idx=None*, *block=None*, ***kwargs*)

**Parameters**

- **expr** (BinaryOp)
- **stmt_idx** (*int | None*)

### angr.analyses.decompiler.peephole_optimizations.shl_to_mul

class angr.analyses.decompiler.peephole_optimizations.shl_to_mul.**ShlToMul**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'a << A => a * (2 ** A)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**

**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.simplify_pc_relative_loads

class angr.analyses.decompiler.peephole_optimizations.simplify_pc_relative_loads.**SimplifyPcRelativeLoads**

Bases: *PeepholeOptimizationExprBase*

Simplifying pc-relative loads.

**NAME = 'Simplify PC-relative loads'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**

**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.single_bit_cond_to_boolexpr

class angr.analyses.decompiler.peephole_optimizations.single_bit_cond_to_boolexpr.**SingleBitCondToBoolExpr**

Bases: *PeepholeOptimizationExprBase*

Convert single-bit conditions to bool expressions

**NAME = 'Convert single-bit conditions to bool expressions (== 0 or == 1)'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.single_bit_xor

**class** angr.analyses.decompiler.peephole_optimizations.single_bit_xor.**SingleBitXor**

Bases: *PeepholeOptimizationExprBase*

**NAME = 'bool_expr ^ 1 => !bool_expr (b)'**

**expr_classes = (<class 'angr.ailment.expression.Convert'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Convert)

### angr.analyses.decompiler.peephole_optimizations.tidy_stack_addr

**class** angr.analyses.decompiler.peephole_optimizations.tidy_stack_addr.**TidyStackAddr**

Bases: *PeepholeOptimizationExprBase*

Consolidate StackBaseOffset objects and constant offsets within each stack address expression.

**NAME = 'Tidy stack addresses'**

**expr_classes = (<class 'angr.ailment.expression.BinaryOp'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (BinaryOp)

### angr.analyses.decompiler.peephole_optimizations.utils

angr.analyses.decompiler.peephole_optimizations.utils.**get_expr_shift_left_amount**(*expr*)

Get the shift amount of a shift-left or multiplication operation if the shift amount is a constant.

**Parameters**
**expr** (*BinaryOp*) – The shift-left or multiplication expression (must be a BinaryOp).

**Return type**
int | None

**Returns**
The shift amount if it is a constant, or None if it is not.

### angr.analyses.decompiler.presets

**class** angr.analyses.decompiler.presets.**DecompilationPreset**

Bases: object

A DecompilationPreset provides a preconfigured set of optimizations and configurations for the Decompiler analysis.

__init__(*name*, *opt_passes*)

**Parameters**

- **name** (*str*)
- **opt_passes** (*list[type[BaseOptimizationPass]]*)

get_optimization_passes(*arch*, *platform*, *additional_opts=None*, *disable_opts=None*)

**Parameters**

- **arch** (*Arch | str*)
- **platform** (*str | None*)

### Submodules

| |
|---|
| *basic* |
| *fast* |
| *full* |
| *malware* |
| *preset* |

### angr.analyses.decompiler.presets.basic

### angr.analyses.decompiler.presets.fast

### angr.analyses.decompiler.presets.full

### angr.analyses.decompiler.presets.malware

### angr.analyses.decompiler.presets.preset

class angr.analyses.decompiler.presets.preset.**DecompilationPreset**

Bases: object

A DecompilationPreset provides a preconfigured set of optimizations and configurations for the Decompiler analysis.

__init__(*name*, *opt_passes*)

**Parameters**

- **name** (*str*)
- **opt_passes** (*list[type[BaseOptimizationPass]]*)

get_optimization_passes(*arch*, *platform*, *additional_opts=None*, *disable_opts=None*)

**Parameters**

- **arch** (*Arch | str*)
- **platform** (*str | None*)

### angr.analyses.decompiler.redundant_label_remover

**class** angr.analyses.decompiler.redundant_label_remover.**RedundantLabelRemover**

Bases: object

Remove redundant labels.

This optimization pass contains two separate passes. The first pass (self._walker0) finds all redundant labels (e.g., two or more labels for the same location) and records the replacement label for redundant labels in self._new_jump_target. The second pass (self._walker1) removes all redundant labels that (a) are not referenced anywhere (determined by jump_targets), or (b) are deemed replaceable by the first pass.

**__init__**(*node*, *jump_targets*)

**Parameters**
**jump_targets** (*set[tuple[int, int | None]]*)

### angr.analyses.decompiler.region_identifier

**type** angr.analyses.decompiler.region_identifier.**TNode** = *Tx*[Block | *MultiNode* | *ConditionNode*]

**type** angr.analyses.decompiler.region_identifier.**TOverlay** = *RegionOverlay*[Block | *MultiNode* | *ConditionNode*]

**type** angr.analyses.decompiler.region_identifier.**TManager** = *OverlayManager*[Block | *MultiNode* | *ConditionNode*]

**type** angr.analyses.decompiler.region_identifier.**TGraph** = networkx.DiGraph[*TNode*]

**class** angr.analyses.decompiler.region_identifier.**RegionIdentifier**

Bases: *Analysis*

A region is a single-entry-single-exit subgraph of control flow. The region identifier recursively identifies the smallest possible regions within a function graph and creates a RegionOverlay object whose nodes are either Blocks or RegionOverlays.

Note, that the analysis may modify the graph in-place. If you want to keep the original graph, set the *update_graph* parameter to False.

**__init__**(*func*, *cond_proc=None*, *graph=None*, *ail_manager=None*, *update_graph=True*, *largest_successor_tree_outside_loop=True*, *force_loop_single_exit=True*, *refine_loops_with_single_successor=False*, *expose_loop_head_backedges=False*, *entry_node_addr=None*)

**test_reducibility**()

**Return type**
bool

### angr.analyses.decompiler.region_overlay

**class** angr.analyses.decompiler.region_overlay.**RegionBound**

Bases: Protocol

The protocol indicating the minimum capabilities of graph nodes to be managed by an overlay region.

**property addr:** int

__init__(**args*, ***kwargs*)

type angr.analyses.decompiler.region_overlay.**Tx** = **U** | *RegionOverlay*[**U**]

angr.analyses.decompiler.region_overlay.**merge_edge_data**(*acc*, *data*)

Merge the data of two complete-graph edges that are about to be merged. `abnormal_entry` merged with an untagged edge keeps the tag and gains `normal_entry`: The jump into the body still becomes a goto later, but the edge must be kept.

**Return type**
dict[str, Any]

**Parameters**

- **acc** (*dict[str, Any]*)
- **data** (*dict[str, Any]*)

class angr.analyses.decompiler.region_overlay.**OverlayManager**

Bases: Generic

OverlayManager owns the single (complete) control-flow graph that all RegionOverlay objects are views of, plus the node-to-innermost-overlay ownership map.

All structural mutations of the complete graph must go through this class (usually via RegionOverlay methods) so that ownership, caches, and the undo log stay consistent.

__init__(*graph*, *expose_loop_head_backedges=False*)

**graph**

**expose_loop_head_backedges**

**root**

**property version: int**

**owner_of**(*node*)

**Return type**
Optional[*RegionOverlay*[TypeVar(T, bound= *RegionBound*)]]

**Parameters**
**node** (*Tx[T]*)

**checkpoint**()

**Return type**
int

**rollback**(*to*)

**Return type**
None

**Parameters**
**to** (*int*)

**commit**(*to*)

Discard undo information past the given checkpoint. If the checkpoint is 0, stop recording entirely.

**Return type**
None

**Parameters**
**to** (*int*)

**graph_add_edge**(*src*, *dst*, ***data*)

**Return type**
None

**graph_strip_entry_tags**(*src*, *dst*)

Drop the abnormal_entry / normal_entry tags of an edge.

**Return type**
None

**graph_remove_edge**(*src*, *dst*)

**Return type**
None

**class** angr.analyses.decompiler.region_overlay.**RegionOverlay**

Bases: Generic

A single-entry region marked over the complete graph held by an OverlayManager. The region tree is built out of RegionOverlay objects (RegionIdentifier emits them) and they are the only region type the decompiler uses.

Overlays form a tree (nested, never overlapping). An overlay's members are either complete-graph nodes it owns directly or child overlays. The region graph and the region graph-with-successors are derived on demand from the complete graph by quotienting child overlays into single nodes; *successors* are likewise derived from edges that cross the region boundary, so they can never go stale.

Read-only API (`head`, `graph`, `graph_with_successors`, `successors`, `cyclic`, `addr`) mirrors the region container consumers expect; `graph`/`graph_with_successors`/`successors` are lazy zero-copy views.

Mutation verbs:

- true structural changes (`add_node`, `remove_node`, `add_edge`, `detach_edge`, `replace_nodes`) pass through to the complete graph, so the effects are immediately visible to all enclosing regions;
- `hide_edge` removes an edge from this overlay's views only (the old "remove it from region graphs but keep the parent edge" pattern);
- `finalize(result_node)` collapses a fully-structured region into a single node of its parent;
- `dissolve()` merges an unsuccessfully-structured region back into its parent.

**__init__**(*mgr*, *head*, *cyclic*, *cyclic_ancestor=False*, *parent=None*)

**Parameters**

- **mgr** (*OverlayManager[T]*)
- **head** (*Tx[T] | None*)
- **cyclic** (*bool*)
- **cyclic_ancestor** (*bool*)
- **parent** (*RegionOverlay | None*)

**head**

**cyclic**

**cyclic_ancestor**

**parent**

**children:** list[*RegionOverlay*]

**edge_marks:** defaultdict[str, set[tuple[*Tx*, *Tx*]]]

**replacement**

**property addr**

**property manager:** *OverlayManager*[T]

**property members:** set[*Tx*[T]]

**ancestors**()

All overlays enclosing this one, including this one itself.

**Return type**
set[*RegionOverlay*]

**underlying_nodes**()

All complete-graph nodes inside this region (including nodes of nested regions).

**Return type**
set[TypeVar(T, bound= *RegionBound*)]

**create_subregion**(*head*, *members*, *cyclic*, *cyclic_ancestor=False*)

Carve a new child overlay out of this overlay. `members` must be a subset of this overlay's members (complete-graph nodes owned by this overlay and/or existing child overlays); `head` must be one of them.

**Return type**
*RegionOverlay*[TypeVar(T, bound= *RegionBound*)]

**Parameters**

- **head** (*Tx[T]*)
- **members** (*Iterable*)
- **cyclic** (*bool*)
- **cyclic_ancestor** (*bool*)

**representative**(*node*)

The node standing for a complete-graph node in this overlay's with-successors view.

**Return type**
*Tx*[TypeVar(T, bound= *RegionBound*)] | None

**Parameters**
**node** (*T*)

**successor_nodes**()

The derived successor set of this region: representatives of all complete-graph nodes targeted by edges leaving the region.

**Return type**
set[*Tx*[TypeVar(T, bound= *RegionBound*)]]

**view_graph**(*full=False*, *include_marked=False*, *blacklisted_edges=frozenset({})*)

A zero-copy, networkx-compatible view of this region (see RegionOverlayGraph).

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**Parameters**

- **full** (*bool*)
- **include_marked** (*bool*)
- **blacklisted_edges** (*frozenset*)

**view**()

The region graph (members only): a zero-copy networkx-compatible view; treat it as read-only.

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**view_with_successors**()

The region graph including successor nodes: a zero-copy view; treat it as read-only.

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**property raw_graph:** *RegionOverlayGraph*[T]

The member view including marked edges (the old graph with cyclic_refinement_outgoing attrs present).

**property raw_graph_with_successors:** *RegionOverlayGraph*[T]

The with-successors view including marked edges.

**property graph**

**property graph_with_successors**

**property successors:** set[*Tx*[T]]

**property full_graph:** None

**complete_graph_entry_count**(*node*)

The number of edges entering a node in the complete graph, counted over every region. For a region object, the entries of its head from outside the region (its own back edges do not count).

**Return type**
int

**Parameters**
**node** (Tx*[T]*)

**external_entry_count**(*node*)

The number of edges entering a node (a successor of this region) from nodes that are not under this region: the entries it keeps once every edge from this region to it has been virtualized.

**Return type**
int

**Parameters**
**node** (Tx*[T]*)

**add_node**(*node*)

Insert a new node into the complete graph as a direct member of this region.

**Return type**
None

**remove_node**(*node*, *absorbed_into=None*, *absorb_out_edges=True*)

Remove a node. If node is a member (or a member overlay's node), it is removed from the complete graph for real. If it is a successor of this region, the removal is interpreted as hiding all edges from this region to it (the successor belongs to an enclosing region and must survive).

When the node's content has been absorbed into another node during structuring, pass that node as absorbed_into: in-edges from outside this region (e.g. abnormal loop entries) are then rewired to it instead of being dropped, so enclosing regions keep their entry edges. With absorb_out_edges (the default), out-edges crossing the region boundary (e.g. loop exits) are rewired to it as well, so the complete graph never loses the region's exit flow; pass False only when the caller re-establishes every successor edge explicitly.

**Return type**
None

**Parameters**

- **node** (*Tx[T]*)
- **absorbed_into** (*Tx[T] | None*)
- **absorb_out_edges** (*bool*)

**hide_edge_to_successor**(*succ*)

**Return type**
None

**underlying_edge_pairs**(*src*, *dst*)

**Return type**
list[tuple[*Tx*[TypeVar(T, bound= *RegionBound*)], *Tx*[TypeVar(T, bound= *RegionBound*)]]]

**Parameters**

- **src** (*Tx[T]*)
- **dst** (*Tx[T]*)

**add_edge**(*src*, *dst*, ***data*)

Add a real edge to the complete graph. Overlay endpoints are resolved to underlying nodes: the destination resolves to its entry (head chain); overlay sources are not supported. Endpoints that are not in the complete graph yet become members of this region (mirroring networkx's implicit node creation).

**Return type**
None

**Parameters**

- **src** (*Tx[T]*)
- **dst** (*Tx[T]*)

**detach_raw_edge**(*u*, *v*)

Remove exactly one complete-graph edge. Unlike detach_edge, other edges between the view nodes representing u and v are kept (e.g. a block that jumps both to a loop's entry and into its body).

**Return type**
None

**Parameters**

- **u** (*T*)
- **v** (*T*)

**detach_edge**(*src*, *dst*)

Remove an edge from the complete graph for real (e.g., when the edge has been virtualized into a goto). Overlay endpoints remove all underlying edges between the two node sets.

**Return type**
None

**Parameters**

- **src** (Tx[*T*])
- **dst** (Tx[*T*])

**mark_edge**(*src*, *dst*, ***attrs*)

Mark a view-level edge (e.g. cyclic_refinement_outgoing) so RegionOverlayGraph hides it by default. Marks live in overlay state, never reach the complete graph, and are remapped/cleared with the region.

**Return type**
None

**Parameters**

- **src** (Tx[*T*])
- **dst** (Tx[*T*])

**absorb_successor_into**(*succ*, *new_node*)

Absorb a successor node into a structured member node in this region's with-successors view only (the successor still belongs to an enclosing region): the successor's view out-edges are re-attached to the member node as view-only extra edges, then the successor disappears from this region's views.

**Return type**
None

**Parameters**

- **succ** (Tx[*T*])
- **new_node** (Tx[*T*])

**drop_edge_marks_from**(*node*, *key*)

Clear marks on all out-edges of a node (the new_node after a replace), undoably.

**Return type**
None

**Parameters**

- **node** (Tx[*T*])
- **key** (*str*)

**remove_edge_with_successors_only**(*src*, *dst*)

Hide an edge from the with-successors view only, leaving the member view and the complete graph alone (a rare asymmetric bookkeeping pattern in Phoenix's switch-case structuring).

**Return type**
None

**Parameters**

- **src** (Tx[T])
- **dst** (Tx[T])

**hide_edge**(*src*, *dst*)

Remove an edge from this overlay's views only. Enclosing regions still see the underlying edge(s).

**Return type**
None

**Parameters**

- **src** (Tx[T])
- **dst** (Tx[T])

**replace_nodes**(*old_node_0*, *new_node*, *old_node_1=None*, *self_loop=True*)

Replace one or two member nodes with a new node, preserving and rewiring all underlying edges (including edges from/to nodes outside this region, which is how results become visible to enclosing regions).

Mirrors StructurerBase.replace_nodes, with one deliberate difference: in-edges of `old_node_1` from outside this region are rewired to `new_node` instead of being dropped (they are entry edges owned by enclosing regions, e.g. abnormal loop entries).

**Return type**
None

**Parameters**

- **old_node_0** (Tx[T])
- **new_node** (Tx[T])
- **old_node_1** (Tx[T] | None)
- **self_loop** (bool)

**snapshot_successors**()

Capture this region's structural successors and how many member edges reach each, taken before the region is structured. finalize() uses it to re-establish the region-to-successor edges that structuring removes when it refines the corresponding control-flow edges into breaks, and to tell successors whose every member edge was virtualized into a goto (see detach_edge()).

**Return type**
dict[Tx[TypeVar(T, bound= RegionBound)], int]

**finalize**(*result_node=None*, *succ_snapshot=None*)

Collapse this fully-structured region into its parent: the region must consist of a single member node (the structuring result), which takes the region's place among the parent's members. Returns that node.

`succ_snapshot` (from snapshot_successors() before structuring) is used to re-establish the edges from result_node to the region's successors: structuring may have removed the live control-flow edges (refining them into breaks), but the structured region still flows to those successors and enclosing regions must see that.

A successor whose every member edge was virtualized into a goto (detach_edge()) is a pure goto target and must NOT be reconnected. Consider a loop with two exits, where `err` is shared with other loops of the function:

**while (1) {**
if (s->bsLive >= 8) break; // exit A: the loop successor if (s->strm->avail_in == 0) goto err;
// exit B: virtualized into a goto by cyclic refinement refill();

} uc = take_bits(); // A …

**err:**
retVal = BZ_DATA_ERROR; goto save_state_and_return;

The loop region has successors A and err. Cyclic refinement keeps A as the loop successor (a break) and turns the exit to err into a goto statement, removing that edge from the complete graph. Reconnecting err here would give the loop node two successors in the parent region, one real and one fictitious: the ITE schema finds no conditions on them, the sequence schema needs a single successor, and last-resort refinement keeps both edges because each target would be orphaned, so the parent region can never be structured. With only A reconnected, the loop node is sequenced with A as usual and err stays reachable through its gotos. Cyclic refinement never virtualizes an exit that is its target's only entry in the complete graph: the region bails and dissolves into its parent, where the target is a member and the acyclic schemas structure it with the loop.

**Parameters**

- **result_node** (*Tx[T] | None*)
- **succ_snapshot** (*set[Tx[T]] | None*)

`collapse_to`(*result_node*)

Collapse this region into its parent by replacing all of its member nodes with a single external result node (the structuring result). Used by structurers that compute their result without destructively reducing the complete graph (e.g. DreamStructurer): the region's members are still present, so their crossing in/out edges are rewired onto `result_node` and the members are removed. Returns result_node.

This is the non-self-collapsing counterpart of finalize(); the legacy GraphRegion path called replace_region() for the same purpose.

**Parameters**
**result_node** (*RegionOverlay[T]*)

`dissolve`()

Merge this region back into its parent (the failure path of structuring): members are reparented, and edges hidden in this region stay hidden in the parent (matching how partially-structured region graphs were merged back before).

**Return type**
None

**class** angr.analyses.decompiler.region_overlay.**RegionOverlayGraph**

Bases: DiGraph, Generic

A read-only, networkx-compatible view of a RegionOverlay that stores no copy of the region's subgraph: all queries traverse the original complete graph through the overlay's membership. Compatible with every networkx algorithm and DiGraph read method because the graph's storage mappings are replaced with lazy atlases.

- `full` selects the with-successors view (the old graph_with_successors) over the member view.
- Edges marked through RegionOverlay.mark_edge (e.g. cyclic_refinement_outgoing) are filtered out by default; pass `all_edges=True` to the query methods, or use with_all_edges(), to include them.
- to_acyclic() derives a view with additional edges blacklisted (e.g. the region head's in-edges), replacing the acyclic graph copies that structuring used to make.

**property full_view:** *RegionOverlayGraph*[T]

The with-successors sibling of this view (zero-copy).

**property member_view:** *RegionOverlayGraph*[T]

The members-only sibling of this view (zero-copy).

**with_all_edges**()

A sibling view that includes edges marked through RegionOverlay.mark_edge.

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**filtered**()

A sibling view that hides edges marked through RegionOverlay.mark_edge (the default).

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**to_acyclic_by_order**(*node_order*)

An acyclic view of this graph, obtained by blacklisting back edges (edges whose source is ordered at or after their destination in `node_order`). Replaces utils.graph.to_acyclic_graph without a copy.

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**Parameters**
**node_order** (*Mapping[Tx[T], int]*)

**to_acyclic**(*blacklisted_edges*)

A new view with the given (view-level) edges additionally blacklisted; used to traverse the region as an acyclic graph without copying it.

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**Parameters**
**blacklisted_edges** (*Sequence[tuple[Tx[T], Tx[T]]]*)

**reverse_view**()

The reversed view of this graph (zero-copy).

**Return type**
*RegionOverlayGraph*[TypeVar(T, bound= *RegionBound*)]

**materialize**()

An independent networkx.DiGraph copy of this view.

**Return type**
networkx.DiGraph[*Tx*[T]]

**edge_marked**(*u*, *v*, *mark_name=None*)

**Return type**
bool

**Parameters**

- **u** (*Tx[T]*)
- **v** (*Tx[T]*)
- **mark_name** (*str | None*)

### angr.analyses.decompiler.region_simplifiers

**class** angr.analyses.decompiler.region_simplifiers.**RegionSimplifier**

Bases: *Analysis*

Simplifies a given region.

**__init__**(*func*, *region*, *ail_manager*, *arg_vvars=None*, *simplify_switches=True*, *simplify_ifelse=True*, *variable_manager=None*, *loopctr_naming=True*)

**Parameters**

- **ail_manager** (*Manager*)
- **arg_vvars** (*set[int] | None*)
- **simplify_switches** (*bool*)
- **simplify_ifelse** (*bool*)
- **variable_manager** (*VariableManagerInternal | None*)
- **loopctr_naming** (*bool*)

### Submodules

| |
|---|
| *cascading_cond_transformer* |
| *cascading_ifs* |
| *expr_folding* |
| *goto* |
| *if_* |
| *ifelse* |
| *loop* |
| *node_address_finder* |
| *region_simplifier* |
| *switch_cluster_simplifier* |
| *switch_expr_simplifier* |

### angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer

**class** angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer.**CascadingConditionTransformer**

Bases: *SequenceWalker*

Identifies and transforms *if { … } else { if { … } else { … } }* to *if { … } else if { … } else if { … }*.

**__init__**(*node*, *manager*)

**Parameters**

**manager** (*Manager*)

### angr.analyses.decompiler.region_simplifiers.cascading_ifs

**class** angr.analyses.decompiler.region_simplifiers.cascading_ifs.**CascadingIfsRemover**

Bases: *SequenceWalker*

Coalesce cascading If constructs. Transforming the following construct:

```
if (cond_a) {
    if (cond_b) {
        true_body
    } else { }
} else { }
```

into:

```
if (cond_a and cond_b) {
    true_body
} else { }
```

**__init__**(*node*, *manager*)

### angr.analyses.decompiler.region_simplifiers.expr_folding

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.**LocationBase**

Bases: object

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.**StatementLocation**

Bases: *LocationBase*

Describes the location of a statement.

**__init__**(*block_addr*, *block_idx*, *stmt_idx*, *phi_stmt=False*)

**Parameters**
**phi_stmt** (*bool*)

**block_addr**

**block_idx**

**stmt_idx**

**phi_stmt**

**copy**()

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionLocation**

Bases: *LocationBase*

Describes the location of an expression.

**__init__**(*block_addr*, *block_idx*, *stmt_idx*, *expr_idx*, *phi_stmt=False*)

**Parameters**
**phi_stmt** (*bool*)

**block_addr**

**block_idx**

**stmt_idx**

**expr_idx**

**phi_stmt**

**statement_location**()

**Return type**
*StatementLocation*

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.**ConditionLocation**

Bases: *LocationBase*

Describes the location of a condition.

**__init__**(*cond_node_addr*, *case_idx=None*)

**Parameters**
**case_idx** (*int | None*)

**node_addr**

**case_idx**

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.**ConditionalBreakLocation**

Bases: *LocationBase*

Describes the location of a conditional break.

**__init__**(*node_addr*)

**node_addr**

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.**LoopNodeFinder**

Bases: *SequenceWalker*

Returns all loops in a given region.

**__init__**(*node*)

**Parameters**
**node** (SequenceNode)

**loop_nodes: list[*LoopNode*]**

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.
**MultiStatementExpressionAssignmentFinder**

Bases: *AILBlockRewriter*

Process statements in MultiStatementExpression objects and find assignments.

**__init__**(*stmt_handler*)

**class** angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionUseFinder**

Bases: *AILBlockRewriter*

Find where each variable is used.

Additionally, determine if the expression being walked has load expressions inside. Such expressions can only be safely folded if there are no Store statements between the expression defining location and its use sites. For example, we can only safely fold variable assignments that use Load() when there are no Store()s between the assignment and its use site. Otherwise, the loaded expression may get updated later by a Store() statement.

Here is a real AIL block:

```
v16 = ((int)v23->field_5) + 1 & 255;
v23->field_5 = ((char)(((int)v23->field_5) + 1 & 255));
v13 = printf("Recieved packet %d for connection with %d\n", v16, a0 & 255);
```

In this case, folding v16 into the last printf() expression would be incorrect, since v23->field_5 is updated by the second statement.

**__init__**()

**uses:** defaultdict[int, set[tuple[*Expression*, *ExpressionLocation* | None]]]

**has_load**

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionCounter**

Bases: *SequenceWalker*

Find all expressions that are assigned once and only used once.

**__init__**(*node*)

**assignments:** defaultdict[Any, set[tuple]]

**outerscope_uses:** dict[int, set[tuple[*Expression*, *LocationBase* | None]]]

**all_uses:** dict[int, set[tuple[*Expression*, *LocationBase* | None]]]

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionSpotter**

Bases: *VVarUsesCollector*

ExpressionSpotter collects uses of vvars and existence of Call expressions.

**__init__**()

**has_calls:** bool

**has_loads:** bool

class angr.analyses.decompiler.region_simplifiers.expr_folding.**InterferenceChecker**

Bases: *SequenceWalker*

Detect for every pair of definition (assignment) - use if there is anything that may interfere with the definition.

Interferences may be caused by:

- another call
- function return
- store statements
- load expressions
- Condition and CascadingCondition nodes

**__init__**(*assignments*, *uses*, *node*, *variable_map*)

**Parameters**

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

**interfered_assignments:** set[int]

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionReplacer**

Bases: *AILBlockRewriter*

**__init__**(*assignments*, *uses*, *variable_map*)

**Parameters**

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

class angr.analyses.decompiler.region_simplifiers.expr_folding.**ExpressionFolder**

Bases: *SequenceWalker*

**__init__**(*assignments*, *uses*, *node*, *variable_map*)

**Parameters**

- **assignments** (*dict[int, Any]*)
- **uses** (*dict[int, Any]*)

class angr.analyses.decompiler.region_simplifiers.expr_folding.**StoreStatementFinder**

Bases: *SequenceWalker*

Determine if there are any Store statements between two given statements.

This class overrides _handle_Sequence() and _handle_MultiNode() to ensure they traverse nodes from top to bottom.

**__init__**(*node*, *intervals*)

**Parameters**
**intervals** (*Iterable[tuple[StatementLocation, LocationBase]]*)

**interval_to_hasstore:** dict[tuple[*StatementLocation*, *StatementLocation*], bool]

**has_store**(*start*, *end*)

**Return type**
bool

**Parameters**

- **start** (StatementLocation)
- **end** (StatementLocation)

### angr.analyses.decompiler.region_simplifiers.goto

class angr.analyses.decompiler.region_simplifiers.goto.**GotoSimplifier**

Bases: *SequenceWalker*

Remove unnecessary Jump statements. This simplifier also has the side effect of detecting Gotos that can't be reduced in the structuring and eventual decompilation output. Because of this, when this analysis is run, gotos in decompilation will be detected and stored in the kb.gotos. See the _handle_irreducible_goto function below.

TODO: Move the recording of Gotos outside this function

**__init__**(*node*, *function=None*, *kb=None*)

### angr.analyses.decompiler.region_simplifiers.if_

class angr.analyses.decompiler.region_simplifiers.if_.**IfSimplifier**

Bases: *SequenceWalker*

Remove unnecessary jump or conditional jump statements if they jump to the successor right afterwards.

**__init__**(*node*)

### angr.analyses.decompiler.region_simplifiers.ifelse

class angr.analyses.decompiler.region_simplifiers.ifelse.**IfElseFlattener**

Bases: *SequenceWalker*

Remove unnecessary else branches and make the else node a direct successor of the previous If node if the If node always returns.

**__init__**(*node*, *functions*)

### angr.analyses.decompiler.region_simplifiers.loop

class angr.analyses.decompiler.region_simplifiers.loop.**LoopSimplifier**

Bases: *SequenceWalker*

Simplifies loops.

**__init__**(*node*, *functions*)

### angr.analyses.decompiler.region_simplifiers.node_address_finder

class angr.analyses.decompiler.region_simplifiers.node_address_finder.**NodeAddressFinder**

Bases: *SequenceWalker*

Walk the entire node and collect all addresses of nodes.

**__init__**(*node*)

### angr.analyses.decompiler.region_simplifiers.region_simplifier

class angr.analyses.decompiler.region_simplifiers.region_simplifier.**RegionSimplifier**

Bases: *Analysis*

Simplifies a given region.

**__init__**(*func*, *region*, *ail_manager*, *arg_vvars=None*, *simplify_switches=True*, *simplify_ifelse=True*, *variable_manager=None*, *loopctr_naming=True*)

**Parameters**

- **ail_manager** (*Manager*)
- **arg_vvars** (*set[int] | None*)
- **simplify_switches** (*bool*)
- **simplify_ifelse** (*bool*)
- **variable_manager** (*VariableManagerInternal | None*)
- **loopctr_naming** (*bool*)

### angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier

**class** angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**CmpOp**

Bases: Enum

All supported comparison operators.

**LT = 0**

**GT = 1**

**EQ = 2**

**NE = 3**

**class** angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**ConditionalRegion**

Bases: object

Describes a conditional region.

**__init__**(*variable*, *op*, *value*, *node*, *parent=None*)

**Parameters**

- **op** (CmpOp)
- **value** (*int*)
- **node** (ConditionNode | *Block*)

**variable**

**op**

**value**

**node**

**parent**

**class** angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**SwitchCaseRegion**

Bases: object

Describes an already-recovered switch region.

**__init__**(*variable*, *node*, *parent=None*)

**Parameters**
**node** (SwitchCaseNode)

**variable**

**node**

**parent**

**class** angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**SwitchClusterFinder**

Bases: *SequenceWalker*

Find comparisons and switches in order to identify switch clusters.

__init__(*node*, *variable_map*)

**var2condnodes:** defaultdict[Any, list[*ConditionalRegion*]]

**var2switches:** defaultdict[Any, list[*SwitchCaseRegion*]]

class angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterReplacer

Bases: *SequenceWalker*

Replace an identified switch cluster with a newly created SwitchCase node.

__init__(*region*, *to_replace*, *replace_with*)

angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**is_simple_jump_node**(*node*, *case_addrs*, *targets=None*)

**Return type**
bool

**Parameters**
**targets** (*set[int] | None*)

angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**filter_cond_regions**(*cond_regions*, *case_addrs*)

Remove all conditional regions that cannot be merged into switch(es).

**Return type**
list[*ConditionalRegion*]

**Parameters**

- **cond_regions** (*list[ConditionalRegion]*)
- **case_addrs** (*set[int]*)

angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**update_switch_case_list**(*cases*, *old_case_id*, *new_case_id*)

Update cases in-place. Make new_case_id directly jump to old_case_id.

**Return type**
None

**Parameters**

- **cases** (*list[tuple[int | tuple[int, ...], SequenceNode]]*)
- **old_case_id** (*int | tuple[int, ...]*)
- **new_case_id** (*int*)

angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**simplify_switch_clusters**(*region*, *var2condnodes*, *var2switches*)

Identify switch clusters and simplify each of them.

**Parameters**

- **region** – The region to simplify.

- **var2condnodes** (dict[Any, list[*ConditionalRegion*]]) – A dict that stores the mapping from (potential) switch variables to conditional regions.
- **var2switches** (dict[Any, list[*SwitchCaseRegion*]]) – A dict that stores the mapping from switch variables to switch-case regions.

**Returns**
None

angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**simplify_lowered_switches**(*region*, *var2cond*, *functions*, *ail_mana*

Identify a lowered switch and simplify it into a switch-case if possible.

**Parameters**

- **region** (*SequenceNode*) – The region to simplify.
- **var2condnodes** (dict[Any, list[*ConditionalRegion*]]) – A dict that stores the mapping from (potential) switch variables to conditional regions.
- **ail_manager** (*Manager*)

**Returns**
None

angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**simplify_lowered_switches_core**(*reg*, *var*, *co*, *fun*, *tio*, *ail*

**Return type**
bool

**Parameters**

- **region** (SequenceNode)
- **ail_manager** (*Manager*)

**class** angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.**FindFirstNodeInSet**

Bases: *SequenceWalker*

Find the first node out of a set of node appearing in a SequenceNode (and its tree).

**__init__**(*node_set*)

**Parameters**
**node_set** (*set[*BaseNode*]*)

### angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier

**class** angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier.**SwitchExpressionSimplifier**

Bases: *SequenceWalker*

Identifies switch expressions that adds or minuses a constant, removes the constant from the switch expression, and adjust all case expressions accordingly.

**__init__**(*node*, *manager*)

**Parameters**
**manager** (*Manager*)

### angr.analyses.decompiler.region_walker

**class** angr.analyses.decompiler.region_walker.**RegionWalker**

Bases: object

A simple traverser class that walks RegionOverlay instances.

**__init__**()

**walk**(*region*)

**Parameters**
**region** (RegionOverlay)

**walk_node**(*region*, *node*)

### angr.analyses.decompiler.return_maker

**class** angr.analyses.decompiler.return_maker.**ReturnMaker**

Bases: *AILGraphWalker*

Traverse the AILBlock graph of a function and update .ret_exprs of all return statements.

**__init__**(*ail_manager*, *arch*, *function*, *ail_graph*)

### angr.analyses.decompiler.semantic_naming

Semantic variable naming patterns for the decompiler.

This package provides various patterns for automatically naming variables based on their semantic usage (loop counters, array indices, etc.).

The naming passes are split into two categories: - Clinic-based passes: Run on the AIL graph before structuring (ClinicNamingBase) - Region-based passes: Run after structuring on the structured region (RegionNamingBase)

Loop counter naming is a Region-based pass that runs in RegionSimplifier to leverage the structured LoopNode information.

**class** angr.analyses.decompiler.semantic_naming.**ArrayIndexNaming**

Bases: *ClinicNamingBase*

Detects variables used as array indices and renames them.

Array index patterns detected: - base + idx * scale (common array indexing) - base + idx (byte array or pointer arithmetic) - ptr[idx] style access

**PRIORITY: int = 50**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for array index patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

**class** angr.analyses.decompiler.semantic_naming.**BooleanNaming**

Bases: *ClinicNamingBase*

Detects variables used as boolean flags and renames them.

Boolean flag patterns detected: - Variables assigned only 0 or 1 - Variables used directly in conditions (if (var) or if (!var)) - Variables compared to 1 - Variables that are results of comparison operations

**PRIORITY: int = 80**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for boolean flag patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

**class** angr.analyses.decompiler.semantic_naming.**CallResultNaming**

Bases: *ClinicNamingBase*

Detects variables that store function call results and names them based on the called function.

**PRIORITY: int = 40**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for function call result patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

**class** angr.analyses.decompiler.semantic_naming.**ClinicNamingBase**

Bases: *SemanticNamingBase*

Base class for semantic naming passes that run in Clinic on AIL graphs.

These passes operate on the raw AIL graph before structuring, allowing them to analyze control flow patterns directly.

**__init__**(*ail_graph*, *variable_manager*, *functions*, *entry_node*, *variable_map*)

**Parameters**

- **ail_graph** (*DiGraph*)
- **variable_manager** (VariableManagerInternal)

- **functions** (FunctionManager)
- **entry_node** (*Block*)
- **variable_map** (VariableMap)

class angr.analyses.decompiler.semantic_naming.**PointerNaming**

Bases: *ClinicNamingBase*

Detects variables used as pointers and renames them.

Pointer patterns detected: - Variables used as addresses in Load/Store operations - Variables involved in pointer arithmetic - Variables passed to functions expecting pointers - Variables receiving pointer-returning function results - Variables used in linked-list patterns (next/prev access)

**PRIORITY: int = 45**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for pointer variable patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

class angr.analyses.decompiler.semantic_naming.**RegionLoopCounterNaming**

Bases: *RegionNamingBase*

Detects loop counter variables from structured LoopNodes and renames them to standard names like i, j, k based on nesting depth.

This pass operates after structuring and uses the iterator information already extracted by LoopSimplifier (stored in LoopNode.iterator).

**PRIORITY: int = 10**

**__init__**(*region*, *variable_manager*, *functions*, *variable_map*)

**Parameters**

- **region** (BaseNode)
- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)

**analyze**()

Analyze the structured region for loop counters and return a mapping of variables to their new names.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name (e.g., “i”, “j”, “k”)

class angr.analyses.decompiler.semantic_naming.**RegionNamingBase**

Bases: *SemanticNamingBase*

Base class for semantic naming passes that run in RegionSimplifier on structured regions.

These passes operate after structuring, allowing them to leverage structured information like LoopNode, ConditionNode, etc. They can reuse loop analysis results from structuring instead of re-analyzing the graph.

**__init__**(*region*, *variable_manager*, *functions*, *variable_map*)

**Parameters**

- **region** (BaseNode)
- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)

**class** angr.analyses.decompiler.semantic_naming.**SemanticNamingBase**

Bases: ABC

Abstract base class for semantic variable naming patterns.

Subclasses implement specific naming patterns (loop counters, array indices, etc.) This is the common base for both Clinic-based and RegionSimplifier-based passes.

**PRIORITY: int = 100**

**__init__**(*variable_manager*, *functions*, *variable_map*)

**Parameters**

- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)
- **variable_map** (VariableMap)

**original_names: dict[*SimVariable*, str | None]**

**abstractmethod analyze**()

Analyze and return a mapping of variables to their suggested names.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to suggested name

**apply_names**(*exclude_vars=None*)

Apply the computed names to the variables.

**Parameters**
**exclude_vars** (set[*SimVariable*] | None) – Variables to skip (already named by higher priority patterns)

**Return type**
set[*SimVariable*]

**Returns**
Set of variables that were renamed

**class** angr.analyses.decompiler.semantic_naming.**SemanticNamingOrchestrator**

Bases: object

Orchestrates multiple semantic naming patterns.

Runs each pattern in priority order (lower PRIORITY value = runs first). Variables named by higher-priority patterns are not renamed by lower-priority ones.

**__init__**(*ail_graph*, *variable_manager*, *functions*, *entry_node*, *variable_map=None*, *patterns=None*)

**Parameters**

- **ail_graph** (*DiGraph*)
- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)
- **entry_node** (*Block*)
- **patterns** (*list[type[ClinicNamingBase]] | None*)

**analyze**()

Run all semantic naming patterns in priority order.

**Return type**
dict[*SimVariable*, str]

**Returns**
Combined mapping of all renamed variables to their new names

**MAX_SHARED_NAME = 3**

**resolve_name_collisions**()

Suffix duplicate names (len, len1, ...) so each variable is unique.

Two distinct variables can get the same name (two strlen -> "len", or CallResultNaming and PointerNaming both -> "ptr").

Once a base name has been used *MAX_SHARED_NAME* times, any further variables that would reuse it are reverted to default (auto-generated) naming: a long run of near-identical names (ptr1 ... ptr19) does not help a reader more than the default names would.

**Return type**
None

**class** angr.analyses.decompiler.semantic_naming.**SizeNaming**

Bases: *ClinicNamingBase*

Detects variables used as sizes, lengths, or counts based on: 1. Being passed as size parameters to known functions 2. Being used in loop bounds (compared with loop counters) 3. Having names that suggest size semantics

**PRIORITY: int = 70**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for size/length variable patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

### Submodules

| | |
|---|---|
| *array_index_naming* | Semantic variable naming for array indices. |
| *boolean_naming* | Semantic variable naming for boolean flag variables. |
| *call_result_naming* | Semantic variable naming for function call results. |
| *naming_base* | Base class for semantic variable naming patterns. |
| *orchestrator* | Orchestrator for semantic variable naming patterns. |
| *pointer_naming* | Semantic variable naming for pointer variables. |
| *region_loop_counter_naming* | Semantic variable naming for loop counters using structured regions. |
| *size_naming* | Semantic variable naming for size/length/count variables. |

### angr.analyses.decompiler.semantic_naming.array_index_naming

Semantic variable naming for array indices.

This module detects variables used as array indices and names them appropriately.

**class** angr.analyses.decompiler.semantic_naming.array_index_naming.**ArrayIndexNaming**

Bases: *ClinicNamingBase*

Detects variables used as array indices and renames them.

Array index patterns detected: - base + idx * scale (common array indexing) - base + idx (byte array or pointer arithmetic) - ptr[idx] style access

**PRIORITY: int = 50**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for array index patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

### angr.analyses.decompiler.semantic_naming.boolean_naming

Semantic variable naming for boolean flag variables.

This module detects variables used as boolean flags and names them appropriately (e.g., found, done, flag, ok).

**class** angr.analyses.decompiler.semantic_naming.boolean_naming.**BooleanNaming**

Bases: *ClinicNamingBase*

Detects variables used as boolean flags and renames them.

Boolean flag patterns detected: - Variables assigned only 0 or 1 - Variables used directly in conditions (if (var) or if (!var)) - Variables compared to 1 - Variables that are results of comparison operations

**PRIORITY: int = 80**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for boolean flag patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

### angr.analyses.decompiler.semantic_naming.call_result_naming

Semantic variable naming for function call results.

This module detects variables that store function call results and names them based on the called function (e.g., malloc result -> ptr, strlen result -> len).

**class** angr.analyses.decompiler.semantic_naming.call_result_naming.**CallResultNaming**

Bases: *ClinicNamingBase*

Detects variables that store function call results and names them based on the called function.

**PRIORITY: int = 40**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for function call result patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

### angr.analyses.decompiler.semantic_naming.naming_base

Base class for semantic variable naming patterns.

This module provides two base classes for semantic naming: - ClinicNamingBase: For passes running in Clinic on AIL graphs - RegionNamingBase: For passes running in RegionSimplifier on structured regions

**class** angr.analyses.decompiler.semantic_naming.naming_base.**SemanticNamingBase**

Bases: ABC

Abstract base class for semantic variable naming patterns.

Subclasses implement specific naming patterns (loop counters, array indices, etc.) This is the common base for both Clinic-based and RegionSimplifier-based passes.

**PRIORITY: int = 100**

**__init__**(*variable_manager*, *functions*, *variable_map*)

**Parameters**

- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)
- **variable_map** (VariableMap)

**original_names: dict[*SimVariable*, str | None]**

**abstractmethod analyze**()

Analyze and return a mapping of variables to their suggested names.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to suggested name

**apply_names**(*exclude_vars=None*)

Apply the computed names to the variables.

**Parameters**
**exclude_vars** (set[*SimVariable*] | None) – Variables to skip (already named by higher priority patterns)

**Return type**
set[*SimVariable*]

**Returns**
Set of variables that were renamed

**class** angr.analyses.decompiler.semantic_naming.naming_base.**ClinicNamingBase**

Bases: *SemanticNamingBase*

Base class for semantic naming passes that run in Clinic on AIL graphs.

These passes operate on the raw AIL graph before structuring, allowing them to analyze control flow patterns directly.

**__init__**(*ail_graph*, *variable_manager*, *functions*, *entry_node*, *variable_map*)

**Parameters**

- **ail_graph** (*DiGraph*)
- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)
- **entry_node** (*Block*)
- **variable_map** (VariableMap)

**class** angr.analyses.decompiler.semantic_naming.naming_base.**RegionNamingBase**

Bases: *SemanticNamingBase*

Base class for semantic naming passes that run in RegionSimplifier on structured regions.

These passes operate after structuring, allowing them to leverage structured information like LoopNode, ConditionNode, etc. They can reuse loop analysis results from structuring instead of re-analyzing the graph.

**__init__**(*region*, *variable_manager*, *functions*, *variable_map*)

**Parameters**

- **region** (BaseNode)
- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)

### angr.analyses.decompiler.semantic_naming.orchestrator

Orchestrator for semantic variable naming patterns.

This module coordinates multiple naming patterns and applies them in priority order.

Note: Loop counter naming is NOT included here as it runs in RegionSimplifier after structuring, where it can leverage the structured LoopNode information.

**class** angr.analyses.decompiler.semantic_naming.orchestrator.**SemanticNamingOrchestrator**

Bases: object

Orchestrates multiple semantic naming patterns.

Runs each pattern in priority order (lower PRIORITY value = runs first). Variables named by higher-priority patterns are not renamed by lower-priority ones.

**__init__**(*ail_graph*, *variable_manager*, *functions*, *entry_node*, *variable_map=None*, *patterns=None*)

**Parameters**

- **ail_graph** (*DiGraph*)
- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)
- **entry_node** (*Block*)
- **patterns** (*list[type[ClinicNamingBase]] | None*)

**renamed_variables:** set[*SimVariable*]

**variable_patterns:** dict[*SimVariable*, str]

**original_names:** dict[*SimVariable*, str | None]

**analyze**()

Run all semantic naming patterns in priority order.

**Return type**
dict[*SimVariable*, str]

**Returns**
Combined mapping of all renamed variables to their new names

**MAX_SHARED_NAME = 3**

**resolve_name_collisions**()

Suffix duplicate names (len, len1, ...) so each variable is unique.

Two distinct variables can get the same name (two strlen -> "len", or CallResultNaming and PointerNaming both -> "ptr").

Once a base name has been used *MAX_SHARED_NAME* times, any further variables that would reuse it are reverted to default (auto-generated) naming: a long run of near-identical names (ptr1 ... ptr19) does not help a reader more than the default names would.

**Return type**
None

### angr.analyses.decompiler.semantic_naming.pointer_naming

Semantic variable naming for pointer variables.

This module detects variables used as pointers and names them appropriately (e.g., ptr, p, cur, next, prev).

**class** angr.analyses.decompiler.semantic_naming.pointer_naming.**PointerNaming**

Bases: *ClinicNamingBase*

Detects variables used as pointers and renames them.

Pointer patterns detected: - Variables used as addresses in Load/Store operations - Variables involved in pointer arithmetic - Variables passed to functions expecting pointers - Variables receiving pointer-returning function results - Variables used in linked-list patterns (next/prev access)

**PRIORITY: int = 45**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for pointer variable patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

### angr.analyses.decompiler.semantic_naming.region_loop_counter_naming

Semantic variable naming for loop counters using structured regions.

This module implements detection and renaming of loop counter variables using the LoopNode structure from the region simplifier. It operates on structured regions after the structuring phase, allowing it to reuse existing loop analysis results instead of re-analyzing the graph.

**class** angr.analyses.decompiler.semantic_naming.region_loop_counter_naming.**RegionLoopCounterNaming**

Bases: *RegionNamingBase*

Detects loop counter variables from structured LoopNodes and renames them to standard names like i, j, k based on nesting depth.

This pass operates after structuring and uses the iterator information already extracted by LoopSimplifier (stored in LoopNode.iterator).

**PRIORITY: int = 10**

**__init__**(*region*, *variable_manager*, *functions*, *variable_map*)

**Parameters**

- **region** (BaseNode)
- **variable_manager** (VariableManagerInternal)
- **functions** (FunctionManager)

**analyze**()

Analyze the structured region for loop counters and return a mapping of variables to their new names.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name (e.g., "i", "j", "k")

### angr.analyses.decompiler.semantic_naming.size_naming

Semantic variable naming for size/length/count variables.

This module detects variables used as sizes, lengths, or counts and names them appropriately (e.g., size, len, count, n).

**class** angr.analyses.decompiler.semantic_naming.size_naming.**SizeNaming**

Bases: *ClinicNamingBase*

Detects variables used as sizes, lengths, or counts based on: 1. Being passed as size parameters to known functions 2. Being used in loop bounds (compared with loop counters) 3. Having names that suggest size semantics

**PRIORITY: int = 70**

**__init__**(**args*, ***kwargs*)

**analyze**()

Analyze the graph for size/length variable patterns.

**Return type**
dict[*SimVariable*, str]

**Returns**
Dictionary mapping SimVariable to new name

### angr.analyses.decompiler.seq_to_blocks

**class** angr.analyses.decompiler.seq_to_blocks.**SequenceToBlocks**

Bases: *SequenceWalker*

A helper class to convert a sequence node into a list of blocks.

**__init__**()

### angr.analyses.decompiler.sequence_walker

**class** angr.analyses.decompiler.sequence_walker.**SequenceWalker**

Bases: object

Walks a SequenceNode and all its nodes, recursively.

**__init__**(*handlers=None*, *exception_on_unsupported=False*, *update_seqnode_in_place=True*, *force_forward_scan=False*)

**Parameters**
**force_forward_scan** (*bool*)

**walk**(*sequence*)

### angr.analyses.decompiler.ssailification

**class** angr.analyses.decompiler.ssailification.**Ssailification**

Bases: *Analysis*

Ssailification (SSA-AIL-ification) transforms an AIL graph to its partial-SSA form.

__init__(*func*, *ail_graph*, *entry=None*, *canonical_size=8*, *stack_pointer_tracker=None*, *func_addr=None*, *ail_manager=None*, *ssa_stackvars=False*, *ssa_tmps=False*, *func_args=None*, *rewrite_vvars=None*, *vvar_id_start=0*)

**Parameters**

- **func** – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **canonical_size** – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.

## Submodules


*consts*
*rewriting*
*rewriting_engine*
*rewriting_state*
*ssailification*
*traversal*
*traversal_engine*
*traversal_state*


### angr.analyses.decompiler.ssailification.consts

### angr.analyses.decompiler.ssailification.rewriting

class angr.analyses.decompiler.ssailification.rewriting.**RewritingAnalysis**

Bases: object

RewritingAnalysis traverses the AIL graph, inserts phi nodes, and rewrites all expression uses to virtual variables when necessary.

__init__(*project*, *func*, *ail_graph*, *phiid_to_udef*, *block_to_phiids*, *rewrite_tmps*, *ail_manager*, *func_args*, *def_to_udef*, *extern_defs*, *incomplete_defs*, *vvar_id_start=0*, *stackvars=False*, *fail_fast=False*)

property max_vvar_id: int | None

create_phi_statements(*node*, *phiid_to_udef*, *phi_ids*)

**Return type**
list[*Assignment*]

**Parameters**

- **node** (*Block*)
- **phiid_to_udef** (*dict[int, UDef]*)
- **phi_ids** (*Iterable[int]*)

static insert_phi_statements(*node*, *phi_stmts*)

**Parameters**

- **node** (*Block*)
- **phi_stmts** (*list[Assignment]*)

### angr.analyses.decompiler.ssailification.rewriting_engine

**class** angr.analyses.decompiler.ssailification.rewriting_engine.**SimEngineSSARewriting**

Bases: *SimEngineNostmtAIL*[*RewritingState*, *Expression* | None, *Statement* | tuple[*Statement*, ...], None]

This engine rewrites every block to insert phi variables and replaces every used variable with their versioned copies at each use location.

**__init__**(*project*, ***, *ail_manager*, *def_to_udef*, *incomplete_defs*, *vvar_id_start=0*, *rewrite_tmps=False*, *stackvars=False*, *fail_fast=False*)

**Parameters**

- **ail_manager** (*Manager*)
- **def_to_udef** (*MutableMapping[Def, UDef]*)
- **incomplete_defs** (*set[Def]*)
- **vvar_id_start** (*int*)
- **rewrite_tmps** (*bool*)
- **stackvars** (*bool*)
- **fail_fast** (*bool*)

**property current_vvar_id: int**

**process**(*state*, ***, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
None

**Parameters**

- **state** (RewritingState)
- **block** (*Block | None*)
- **whitelist** (*set[int] | None*)

### angr.analyses.decompiler.ssailification.rewriting_state

**class** angr.analyses.decompiler.ssailification.rewriting_state.**RewritingState**

Bases: object

The abstract state for the expression rewriting engine.

**__init__**(*loc*, *arch*, *func*, *original_block*, *registers=None*, *stackvars=None*)

**Parameters**

- **loc** (AILCodeLocation)
- **original_block** (*Block*)
- **registers** (*dict[int, VirtualVariable] | None*)
- **stackvars** (*ChainMapCOW[int, VirtualVariable] | None*)

**copy**()

**Return type**
*RewritingState*

**append_statement**(*stmt*)

**Parameters**
**stmt** (Statement)

### angr.analyses.decompiler.ssailification.ssailification

**type** angr.analyses.decompiler.ssailification.ssailification.**Kind** = Literal['stack', 'reg']

**type** angr.analyses.decompiler.ssailification.ssailification.**UDef** = tuple[*Kind*, int, int]

**type** angr.analyses.decompiler.ssailification.ssailification.**Def** = *StackBaseOffset* | *Register*

**class** angr.analyses.decompiler.ssailification.ssailification.**Ssailification**

Bases: *Analysis*

Ssailification (SSA-AIL-ification) transforms an AIL graph to its partial-SSA form.

**__init__**(*func*, *ail_graph*, *entry=None*, *canonical_size=8*, *stack_pointer_tracker=None*, *func_addr=None*, *ail_manager=None*, *ssa_stackvars=False*, *ssa_tmps=False*, *func_args=None*, *rewrite_vvars=None*, *vvar_id_start=0*)

**Parameters**

- **func** – The subject of the analysis: a function, or a single basic block
- **ail_graph** – The AIL graph to transform.
- **canonical_size** – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.

### angr.analyses.decompiler.ssailification.traversal

**class** angr.analyses.decompiler.ssailification.traversal.**TraversalAnalysis**

Bases: object

TraversalAnalysis traverses the AIL graph and collects definitions.

**__init__**(*project*, *func*, *ail_graph*, *sp_tracker*, *bp_as_gpr*, *stackvars*, *tmps*, *func_args*, *functions*, *variable_map=None*)

### angr.analyses.decompiler.ssailification.traversal_engine

angr.analyses.decompiler.ssailification.traversal_engine.**offset_sort_key**(*v*)

**Return type**
tuple[int, int, int, int]

**Parameters**
**v** (*tuple[int | None, int]*)

class angr.analyses.decompiler.ssailification.traversal_engine.**DefInfo**

Bases: object

Information about a def, or an expression that defines a new SSA variable.

**def_:** *Def*

**kind:** *Kind*

**loc:** *AILCodeLocation*

**variable_offset:** int

**variable_size:** int

**store_offset:** int

**store_size:** int

**supercedes:** set[*Def*]

property variable_range

property variable_endoffset

__init__(*def_*, *kind*, *loc*, *variable_offset*, *variable_size*, *store_offset*, *store_size*, *supercedes*)

**Parameters**

- **def_** (Def)
- **kind** (Kind)
- **loc** (AILCodeLocation)
- **variable_offset** (*int*)
- **variable_size** (*int*)
- **store_offset** (*int*)
- **store_size** (*int*)
- **supercedes** (*set[Def]*)

**Return type**
None

class angr.analyses.decompiler.ssailification.traversal_engine.**SimEngineSSATraversal**

Bases: *SimEngineLightAIL*[*TraversalState*, *Value*, None, None]

This engine collects all register and stack variable locations and links them to the block of their creation.

__init__(*project*, *simos*, *sp_tracker=None*, *bp_as_gpr=False*, *stackvars=False*, *use_tmps=False*, *functions=None*, *variable_map=None*)

**Parameters**

- **project** (Project)
- **bp_as_gpr** (*bool*)
- **stackvars** (*bool*)
- **use_tmps** (*bool*)

- **functions** (*Callable[[int | str], Function | None] | None*)

**variable_map**: *VariableMap* | None

**def_info**: dict[*Def*, *DefInfo*]

**pending_ptr_defines_nonlocal**: dict[int, tuple[*AILCodeLocation*, *StackBaseOffset*, set[tuple[int, int]], bool, bool]]

**hclb_side_exit_state**: *TraversalState* | None

**finalize**()

**perform_def**(*kind*, *def_*, *variable_offset*, *variable_size*, *store_offset*, *store_size*, *loc=None*, *supercedes=None*)

**Parameters**

- **kind** (Kind)
- **def_** (Def)
- **variable_offset** (*int*)
- **variable_size** (*int*)
- **store_offset** (*int*)
- **store_size** (*int*)
- **loc** (AILCodeLocation | *None*)
- **supercedes** (*Iterable[Def] | None*)

**stackvar_get**(*base_offset*, *extra_offset*, *base_size*)

**Return type**
*Value*

**Parameters**

- **base_offset** (*int*)
- **extra_offset** (*int*)
- **base_size** (*int*)

**stackvar_set**(*base_offset*, *extra_offset*, *base_size*, *value*)

**Parameters**

- **base_offset** (*int*)
- **extra_offset** (*int*)
- **base_size** (*int*)
- **value** (Value)

**register_get**(*offset*, *size*, *def_*)

**Return type**
*Value*

**Parameters**

- **offset** (*int*)

- **size** (*int*)
- **def_** (*Def | None*)

**register_set**(*offset*, *size*, *value*, *def_*)

**Parameters**

- **offset** (*int*)
- **size** (*int*)
- **value** (Value)
- **def_** (Def)

### angr.analyses.decompiler.ssailification.traversal_state

**type** angr.analyses.decompiler.ssailification.traversal_state.**Value** = **set**[**tuple**[**int** | **None**, **int**]]

angr.analyses.decompiler.ssailification.traversal_state.**has_conflicting_value_types**(*vs*)

Value contains two types of entries: `(int, *)` that indicates a stack offset, and `(None, int)` that indicates a constant value. This method returns True if a set of Values contains both types of entries, otherwise False.

**Return type**
bool

**Parameters**
**vs** (Value)

**class** angr.analyses.decompiler.ssailification.traversal_state.**TraversalState**

Bases: object

The abstract state for the traversal engine.

**__init__**(*arch*, *func*, *live_registers=None*, *live_stackvars=None*, *register_blackout=None*, *live_vvars=None*, *stackvar_bases=None*, *register_bases=None*, *stackvar_defs=None*, *register_defs=None*, *pending_ptr_defines_nonlocal_live=None*)

**Parameters**

- **live_registers** (*MutableMapping[int, Value] | None*)
- **live_stackvars** (DefaultChainMapCOW[*int*, Value] | *None*)
- **register_blackout** (*set[int] | None*)
- **live_vvars** (DefaultChainMapCOW[*int*, Value] | *None*)
- **stackvar_bases** (ChainMapCOW[*int*, *tuple[int, int]*] | *None*)
- **register_bases** (*MutableMapping[int, tuple[int, int]] | None*)
- **stackvar_defs** (DefaultChainMapCOW[*int*, *set[Def]*] | *None*)
- **register_defs** (*MutableMapping[int, set[Def]] | None*)
- **pending_ptr_defines_nonlocal_live** (*set[int] | None*)

**stackvar_unify**(*offset*, *size*)

**Return type**
tuple[int, int, set[int]]

**Parameters**

- **offset** (*int*)
- **size** (*int*)

**register_unify**(*offset*, *size*)

**Return type**
tuple[int, int, set[int]]

**Parameters**

- **offset** (*int*)
- **size** (*int*)

**copy**()

**Return type**
*TraversalState*

**merge**(**others*)

**Return type**
bool

**Parameters**
**others** (TraversalState)

### angr.analyses.decompiler.stack_item

**class** angr.analyses.decompiler.stack_item.**StackItemType**

Bases: Enum

Enum for the type of stack items.

**UNKNOWN = 0**

**SAVED_BP = 1**

**SAVED_REGS = 2**

**ARGUMENT = 3**

**RET_ADDR = 4**

**STACK_CANARY = 5**

**class** angr.analyses.decompiler.stack_item.**StackItem**

Bases: object

A stack item describes a piece of data that is stored on the stack at a certain offset (usually negative).

**__init__**(*offset*, *size*, *name*, *item_type=StackItemType.UNKNOWN*)

**Parameters**

- **offset** (*int*)
- **size** (*int*)
- **name** (*str*)
- **item_type** (StackItemType)

**offset:** *int*

**size:** *int*

**name:** *str*

**item_type:** *StackItemType*

### angr.analyses.decompiler.stackarg_offset_manager

**class** angr.analyses.decompiler.stackarg_offset_manager.**StackArgOffsetManager**

Bases: object

A manager that keeps track of stack argument offsets and sizes for all call sites in a function. This manager also keeps track of the stack virtual variables that can be eliminated for each call site because they are stack arguments.

*all_stackarg_vvars* and *stackoff_to_vvars* map stack arguments to virtual variable IDs. You must call *update_stackoff_vvars()* to populate these two attributes after Stage 1 SSA rewriting. *is_stackarg_vvar()* is only valid after *update_stackoff_vvars()* is called.

**__init__**(*bits*)

**Parameters**
**bits** (*int*)

**add_call_stack_arg_offset**(*block_addr*, *block_idx*, *ins_addr*, *stack_arg_offset*, *stack_arg_size*)

**Parameters**

- **block_addr** (*int*)
- **block_idx** (*int | None*)
- **ins_addr** (*int*)
- **stack_arg_offset** (*int*)
- **stack_arg_size** (*int*)

**merge**(*other*)

**Parameters**
**other** (StackArgOffsetManager)

**get_stackarg_offsets**()

**Return type**
set[int]

**get_stackarg_insaddrs**()

**Return type**
set[int]

**update_stackoff_vvars**(*rd*)

**Return type**
None

**Parameters**
**rd** (SRDAModel)

**is_stackarg_vvar**(*vvar_id*)

**Return type**
bool

**Parameters**
**vvar_id** (*int*)

### angr.analyses.decompiler.stl_field_accessors

Named accessors for fields of the C++ STL classes registered in the `cpp::std` `SimTypeCollection`.

The KnownPattern machinery (see *angr.analyses.decompiler.known_patterns*) recognizes inlined STL idioms structurally, before variable recovery, and outlines them into calls such as `std::string::length(s)`. That approach has a hard floor: the smallest accessors compile to a single word-sized load at a fixed offset. `std::string::c_str()` on libstdc++ is exactly `mov (%rcx), %rcx`, byte-identical to `std::vector<T>::data()`, `unique_ptr::get()`, and every other pointer-field dereference in the program. No structural pattern can tell them apart, and one that tried would fire on unrelated code constantly.

The information that does distinguish them only exists later: once an anchor pattern (`length/capacity/empty/...`) has fired anywhere in the function, Typehoon types the container pointer as the corresponding `cpp::std` class, and every other access to it is resolved to a named field. Naming those accesses after their accessors adds no guess of its own. It inherits exactly the confidence of the recovered type, so it can never fire on a pointer that type inference did not already call an STL container, and it does inherit a wrong type: where an anchor pattern mistyped the container, the accessor name is wrong in the same way the field name was.

This module holds the field -> accessor table used to name those already-typed accesses. Only fields whose value is the accessor's return value are listed:

- `m_capacity` is libstdc++'s `_M_allocated_capacity`, which is the active member of a union only for heap-allocated strings, so a raw read of it is not `capacity()`.
- `m_end_of_storage` has no accessor of its own (`capacity()` is a subtraction), so it is not listed either.

angr.analyses.decompiler.stl_field_accessors.**stl_accessor_name**(*ty*, *field_name*)

Return the fully qualified accessor name for reading `field_name` out of `ty`, e.g. `"std::string::c_str"` for the `m_data` field of `std::string`, or `None` when `ty` is not a registered `cpp::std` class or the field has no equivalent accessor.

The lookup is keyed on `SimCppClass.unique_name` and additionally requires that name to be registered in the `cpp::std` type collection, so a user-defined struct that happens to have a field called `m_data` can never match.

**Return type**
str | None

**Parameters**
- **ty** (SimType)
- **field_name** (*str*)

### angr.analyses.decompiler.structured_codegen

**class** angr.analyses.decompiler.structured_codegen.**BaseStructuredCodeGenerator**

Bases: object

**__init__**(*flavor=None*, *notes=None*, *expr_comments=None*, *stmt_comments=None*, *const_formats=None*)

**map_ast_to_pos:** dict[*SimVariable*, set[*PositionMappingElement*]] | None

**expr_comments:** dict[int, str]

**stmt_comments:** dict[int, str]

**const_formats:** dict[tuple[int, int, str], dict[str, bool]]

**version:** str

**timestamp:** int

**ident_counters:** dict[str, count]

**static adjust_mapping_positions**(*offset*, *pos_to_node*, *pos_to_addr*, *addr_to_pos*)

Adjust positions in the mappings to account for the notes that are prepended to the text.

**Parameters**

- **offset** (int) – The length of the notes to prepend.
- **pos_to_node** (*PositionMapping*) – The position to node mapping.
- **pos_to_addr** (*PositionMapping*) – The position to address mapping.
- **addr_to_pos** (*InstructionMapping*) – The address to position mapping.

**Return type**
tuple[*PositionMapping*, *PositionMapping*, *InstructionMapping*]

**Returns**
Adjusted mappings.

**reapply_options**(*options*)

**regenerate_text**()

**Return type**
None

**reload_variable_types**()

**Return type**
None

**next_ident**(*key*)

**Return type**
str

**Parameters**
**key** (*str*)

**reset_ident_counters**()

**Return type**
None

**next_node_idx**()

**Return type**
int

**class** angr.analyses.decompiler.structured_codegen.**CStructuredCodeGenerator**

Bases: *BaseStructuredCodeGenerator*, *Analysis*, *Serializable*

**__init__**(*func*, *sequence*, *indent=0*, *cfg=None*, *func_args=None*, *binop_depth_cutoff=16*, *show_casts=True*, *braces_on_own_lines=True*, *use_compound_assignments=True*, *show_local_types=True*, *comment_gotos=False*, *cstyle_null_cmp=True*, *flavor=None*, *stmt_comments=None*, *expr_comments=None*, *show_externs=True*, *externs=None*, *const_formats=None*, *show_demangled_name=True*, *show_disambiguated_name=True*, *ail_graph=None*, *simplify_else_scope=True*, *cstyle_ifs=True*, *omit_func_header=False*, *display_block_addrs=False*, *display_vvar_ids=False*, *min_data_addr=4194304*, *notes=None*, *display_notes=True*, *max_str_len=None*, *prettify_thiscall=False*, *cstyle_void_param=True*, *indent_size=4*, *variable_map=None*, *stl_accessor_calls=False*)

**Parameters**

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)
- **min_data_addr** (*int*)
- **display_notes** (*bool*)
- **max_str_len** (*int | None*)
- **prettify_thiscall** (*bool*)
- **cstyle_void_param** (*bool*)
- **indent_size** (*int*)
- **variable_map** (*VariableMap | None*)
- **stl_accessor_calls** (*bool*)

**ailexpr2cnode:** dict[tuple[*Expression*, bool], *CExpression*] | None

**cnode2ailexpr:** dict[*CExpression*, *Expression*] | None

**map_ast_to_pos:** dict[*SimVariable*, set[*PositionMappingElement*]] | None

**map_addr_to_label:** dict[tuple[int, int | None], *CLabel*]

**cfunc:** *CFunction* | None

**cexterns:** set[*CVariable*] | None

**reapply_options**(*options*)

**cleanup**()

Remove existing rendering results.

**regenerate_text**()

Re-render text and re-generate all sorts of mapping information.

**Return type**
None

**render_text**(*cfunc*)

**Return type**
*RenderResult*

**Parameters**
**cfunc** (*CFunction*)

**render_notes**()

Render decompilation notes.

**Return type**
str

**Returns**
A string containing all notes.

**reload_variable_types**()

**Return type**
None

**default_simtype_from_bits**(*n*, *signed=True*)

**Return type**
*SimType*

**Parameters**

- **n** (*int*)
- **signed** (*bool*)

**variables_unify**(*v1*, *v2*)

**Return type**
bool

**Parameters**

- **v1** (VirtualVariable)
- **v2** (VirtualVariable)

**ERRNO_LOCATION_FUNCS = frozenset({'__errno', '__errno_location', '__error', '_errno'})**

The libc functions that return &errno. errno is a macro that dereferences one of them, so it never survives into a binary as a symbol; what reaches decompilation is *(__errno_location()), which is both correct and unreadable.

**class** angr.analyses.decompiler.structured_codegen.**CStructuredCodeWalker**

Bases: object

**handle**(*obj*)

**handle_default**(*obj*)

**handle_CFunction**(*obj*)

**handle_CStatements**(*obj*)

**handle_CWhileLoop**(*obj*)

**handle_CDoWhileLoop**(*obj*)

**handle_CForLoop**(*obj*)

**handle_CIfElse**(*obj*)

**handle_CIfBreak**(*obj*)

**handle_CSwitchCase**(*obj*)

**handle_CAssignment**(*obj*)

**handle_CExpressionStatement**(*obj*)

**handle_CFunctionCall**(*obj*)

**handle_CReturn**(*obj*)

**handle_CGoto**(*obj*)

**handle_CIndexedVariable**(*obj*)

**handle_CVariableField**(*obj*)

**handle_CUnaryOp**(*obj*)

**handle_CBinaryOp**(*obj*)

**handle_CTypeCast**(*obj*)

**handle_CITE**(*obj*)

**handle_CVectorConvert**(*obj*)

**handle_CVEXCCallExpression**(*obj*)

**class** angr.analyses.decompiler.structured_codegen.**DummyStructuredCodeGenerator**

Bases: *BaseStructuredCodeGenerator*, *Serializable*

A dummy structured code generator that only stores user-specified information.

**__init__**(*flavor*, *expr_comments=None*, *stmt_comments=None*, *configuration=None*, *const_formats=None*)

**Parameters**
**flavor** (*str*)

**class** angr.analyses.decompiler.structured_codegen.**ImportSourceCode**

Bases: *BaseStructuredCodeGenerator*, *Analysis*

**__init__**(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

**regenerate_text**()

**class** angr.analyses.decompiler.structured_codegen.**InstructionMapping**

Bases: object

**__init__**()

**items**()

**values**()

**add_mapping**(*ins_addr*, *posmap_pos*)

**get_nearest_pos**(*ins_addr*)

**Return type**
int | None

**Parameters**
**ins_addr** (*int*)

class angr.analyses.decompiler.structured_codegen.**InstructionMappingElement**

Bases: object

**__init__**(*ins_addr*, *posmap_pos*)

**ins_addr:** int

**posmap_pos:** int

class angr.analyses.decompiler.structured_codegen.**PositionMapping**

Bases: object

**DUPLICATION_CHECK = True**

**__init__**()

**items**()

**values**()

**add_mapping**(*start_pos*, *length*, *obj*)

**get_node**(*pos*)

**Parameters**
**pos** (*int*)

**get_element**(*pos*)

**Return type**
*PositionMappingElement* | None

**Parameters**
**pos** (*int*)

class angr.analyses.decompiler.structured_codegen.**PositionMappingElement**

Bases: object

**__init__**(*start*, *length*, *obj*)

**start:** int

**length:** int

**obj**

class angr.analyses.decompiler.structured_codegen.**RustStructuredCodeGenerator**

Bases: *BaseStructuredCodeGenerator*, *Analysis*

**__init__**(*func*, *sequence*, *indent=0*, *cfg=None*, *func_args=None*, *binop_depth_cutoff=16*, *show_casts=True*, *braces_on_own_lines=False*, *use_compound_assignments=True*, *show_local_types=False*, *comment_gotos=False*, *cstyle_null_cmp=True*, *flavor=None*, *stmt_comments=None*, *expr_comments=None*, *show_externs=False*, *externs=None*, *const_formats=None*, *show_demangled_name=True*, *show_disambiguated_name=True*, *ail_graph=None*, *simplify_else_scope=True*, *cstyle_ifs=True*, *omit_func_header=False*, *display_block_addrs=False*, *display_vvar_ids=False*, *min_data_addr=4194304*, *notes=None*, *display_notes=True*, *max_str_len=None*, *prettify_thiscall=False*, *cstyle_void_param=True*, *indent_size=4*, *variable_map=None*, *stl_accessor_calls=False*)

**Parameters**

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)
- **min_data_addr** (*int*)
- **display_notes** (*bool*)
- **max_str_len** (*int | None*)
- **prettify_thiscall** (*bool*)
- **cstyle_void_param** (*bool*)
- **indent_size** (*int*)
- **variable_map** (VariableMap | *None*)
- **stl_accessor_calls** (*bool*)

**reapply_options**(*options*)

**cleanup**()

Remove existing rendering results.

**regenerate_text**()

Re-render text and re-generate all sorts of mapping information.

**Return type**
None

**RENDER_TYPE**

alias of tuple[str, *PositionMapping*, *PositionMapping*, *InstructionMapping*, dict[Any, set[Any]]]

**render_text**(*rust_func*)

**Return type**
RENDER_TYPE

**Parameters**
**rust_func** (RustFunction)

**render_notes**()

Render decompilation notes.

**Return type**
str

**Returns**
A string containing all notes.

**reload_variable_types**()

**Return type**
None

**default_simtype_from_size**(*n*, *signed=True*)

**Return type**
*RustSimTypeInt*

**Parameters**

- **n** (*int*)

- **signed** (*bool*)

### Submodules

| | |
|---|---|
| *base* | |
| *c* | |
| *c_serialize* | Protobuf serialization helpers for the C AST defined in c. |
| *dummy* | |
| *dwarf_import* | |
| *rust* | |

### angr.analyses.decompiler.structured_codegen.base

**class** angr.analyses.decompiler.structured_codegen.base.**CConstantType**

Bases: Enum

Describes the value type of a CConstant class.

**INT = 0**

**FLOAT = 1**

**STRING = 2**

**class** angr.analyses.decompiler.structured_codegen.base.**PositionMappingElement**

Bases: object

**__init__**(*start*, *length*, *obj*)

**start: int**

**length: int**

**obj**

**class** angr.analyses.decompiler.structured_codegen.base.**PositionMapping**

Bases: object

**DUPLICATION_CHECK = True**

**__init__**()

**items**()

**values**()

**add_mapping**(*start_pos*, *length*, *obj*)

**get_node**(*pos*)

**Parameters**
**pos** (*int*)

**get_element**(*pos*)

**Return type**
*PositionMappingElement* | None

**Parameters**
**pos** (*int*)

class angr.analyses.decompiler.structured_codegen.base.**InstructionMappingElement**

Bases: object

**__init__**(*ins_addr*, *posmap_pos*)

**ins_addr**: int

**posmap_pos**: int

class angr.analyses.decompiler.structured_codegen.base.**InstructionMapping**

Bases: object

**__init__**()

**items**()

**values**()

**add_mapping**(*ins_addr*, *posmap_pos*)

**get_nearest_pos**(*ins_addr*)

**Return type**
int | None

**Parameters**
**ins_addr** (*int*)

class angr.analyses.decompiler.structured_codegen.base.**BaseStructuredCodeGenerator**

Bases: object

**__init__**(*flavor=None*, *notes=None*, *expr_comments=None*, *stmt_comments=None*, *const_formats=None*)

**map_ast_to_pos**: dict[*SimVariable*, set[*PositionMappingElement*]] | None

**expr_comments**: dict[int, str]

**stmt_comments**: dict[int, str]

**const_formats**: dict[tuple[int, int, str], dict[str, bool]]

**version**: str

**timestamp**: int

**ident_counters**: dict[str, count]

**static adjust_mapping_positions**(*offset*, *pos_to_node*, *pos_to_addr*, *addr_to_pos*)

Adjust positions in the mappings to account for the notes that are prepended to the text.

**Parameters**

- **offset** (int) – The length of the notes to prepend.
- **pos_to_node** (*PositionMapping*) – The position to node mapping.

- **pos_to_addr** (*PositionMapping*) – The position to address mapping.
- **addr_to_pos** (*InstructionMapping*) – The address to position mapping.

**Return type**
tuple[*PositionMapping*, *PositionMapping*, *InstructionMapping*]

**Returns**
Adjusted mappings.

**reapply_options**(*options*)

**regenerate_text**()

**Return type**
None

**reload_variable_types**()

**Return type**
None

**next_ident**(*key*)

**Return type**
str

**Parameters**
**key** (*str*)

**reset_ident_counters**()

**Return type**
None

**next_node_idx**()

**Return type**
int

angr.analyses.decompiler.structured_codegen.base.**vector_convert_name**(*expr*)

Intrinsic-style name for a lane-wise conversion, e.g. `ConvF32toI32Sx4` for `Iop_F32toI32Sx4`.

**Return type**
str

**Parameters**
**expr** (Convert)

### angr.analyses.decompiler.structured_codegen.c

**type** angr.analyses.decompiler.structured_codegen.c.**RenderResult** = tuple[str, *PositionMapping*, *PositionMapping*, *InstructionMapping*, dict[Any, set[Any]]]

angr.analyses.decompiler.structured_codegen.c.**qualifies_for_simple_cast**(*ty1*, *ty2*)

angr.analyses.decompiler.structured_codegen.c.**qualifies_for_width_cast**(*ty*)

angr.analyses.decompiler.structured_codegen.c.**qualifies_for_implicit_cast**(*ty1*, *ty2*)

angr.analyses.decompiler.structured_codegen.c.**extract_terms**(*expr*)

**Return type**
tuple[int, list[tuple[int, *CExpression*]]]

**Parameters**
**expr** (CExpression)

angr.analyses.decompiler.structured_codegen.c.**is_machine_word_size_type**(*type_*, *arch*)

**Return type**
bool

**Parameters**

- **type_** (SimType)
- **arch** (*Arch*)

angr.analyses.decompiler.structured_codegen.c.**guess_value_type**(*value*, *project*)

**Return type**
*SimType* | None

**Parameters**

- **value** (*int*)
- **project** (Project)

angr.analyses.decompiler.structured_codegen.c.**type_equals**(*t0*, *t1*)

**Return type**
bool

**Parameters**

- **t0** (SimType)
- **t1** (SimType)

angr.analyses.decompiler.structured_codegen.c.**type_layout_key**(*ty*, *memo=None*)

A structural sort key for a type, derived purely from its memory layout (sizes, field offsets, and the layouts of field/element/pointee types) and not from any user-renamable struct or field name. This lets the code generator order type definitions stably without their order changing when the user renames a struct or a field.

The key covers the type down to TYPE_LAYOUT_KEY_DEPTH levels of struct nesting. The key summarizes the member of each struct level in a hash.

**Return type**
str

**Parameters**
**memo** (*_LayoutKeyMemo | None*)

angr.analyses.decompiler.structured_codegen.c.**cextern_sort_key**(*cextern*)

A stable sort key for extern variables, based on the variable's address. Unlike the variable name, the address does not change when the user renames the variable, so the ordering of extern definitions stays put across renames.

**Return type**
tuple

angr.analyses.decompiler.structured_codegen.c.**type_to_c_repr_chunks**(*ty*, *name=None*, *name_type=None*, *full=False*, *indent_str=''*, *indent_delta=4*, *memo=None*)

Helper generator function to turn a SimType into generated tuples of (C-string, AST node).

**Parameters**

- **indent_delta** (int) – Number of space characters used to indent each struct field one level deeper.
- **memo** (set[int]|None) – IDs of the aggregates currently being rendered further up the stack, so that a self-referential type is elided instead of recursed into. An aggregate is added before its members are rendered and removed once they are done.
- **ty** (SimType)

**class** angr.analyses.decompiler.structured_codegen.c.**CConstruct**

Bases: object

Represents a program construct in C. Acts as the base class for all other representation constructions.

**__init__**(*codegen*, *tags=None*)

**tags**

**codegen:** *CStructuredCodeGenerator*

**ident:** str

**idx:** int

**c_repr**(*initial_pos=0*, *indent=0*, *pos_to_node=None*, *pos_to_addr=None*, *addr_to_pos=None*)

Creates the C representation of the code and displays it by constructing a large string. This function is called by each program function that needs to be decompiled. The map_pos_to_node and map_pos_to_addr act as position maps for the location of each variable and statement to be tracked for later GUI operations. The map_pos_to_addr also contains expressions that are nested inside of statements.

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**static indent_str**(*indent=0*)

**class** angr.analyses.decompiler.structured_codegen.c.**CFunction**

Bases: *CConstruct*

Represents a function in C.

**__init__**(*addr*, *name*, *functy*, *arg_list*, *statements*, *variables_in_use*, *variable_manager*, *demangled_name=None*, *show_demangled_name=True*, *omit_header=False*, ***kwargs*)

**Parameters**

- **functy** (SimTypeFunction)
- **arg_list** (*list*[CVariable])

**addr**

**name**

**functy**

**arg_list**

**statements**

**variables_in_use**

**variable_manager:** *VariableManagerInternal*

**demangled_name**

**unified_local_vars:** dict[*SimVariable*, set[tuple[*CVariable*, *SimType*]]]

**show_demangled_name**

**omit_header**

**refresh**()

**get_unified_local_vars**()

**Return type**
dict[*SimVariable*, set[tuple[*CVariable*, *SimType*]]]

**variable_list_repr_chunks**(*indent=0*)

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**headerless_c_repr_chunks**(*indent=0*)

**full_c_repr_chunks**(*indent=0*, *asexpr=False*)

**static sort_local_vars**(*local_vars*)

**Return type**
list[*SimVariable*]

**Parameters**
**local_vars** (*Iterable[SimVariable]*)

**class** angr.analyses.decompiler.structured_codegen.c.**CStatement**

Bases: *CConstruct*

Represents a statement in C.

**__init__**(*tags=None*, ***, *codegen*)

**codegen:** *CStructuredCodeGenerator*

**ident:** str

**idx:** int

**tags**

**class** angr.analyses.decompiler.structured_codegen.c.**CExpression**

Bases: *CConstruct*

Base class for C expressions.

**__init__**(*collapsed=False*, *tags=None*, ***, *codegen*)

**collapsed**

**property type:** *SimType* | None

**set_type**(*v*)

class angr.analyses.decompiler.structured_codegen.c.**CStatements**

Bases: *CStatement*

Represents a sequence of statements in C.

**__init__**(*statements*, *addr=None*, ***kwargs*)

**statements**

**addr**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CAILBlock**

Bases: *CStatement*

Represents a block of AIL statements.

**__init__**(*block*, ***kwargs*)

**block**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CLoop**

Bases: *CStatement*

Represents a loop in C.

class angr.analyses.decompiler.structured_codegen.c.**CWhileLoop**

Bases: *CLoop*

Represents a while loop in C.

**__init__**(*condition*, *body*, ***kwargs*)

**condition**

**body**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CDoWhileLoop**

Bases: *CLoop*

Represents a do-while loop in C.

**__init__**(*condition*, *body*, ***kwargs*)

**condition**

**body**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CForLoop**

Bases: *CStatement*

Represents a for-loop in C.

**__init__**(*initializer*, *condition*, *iterator*, *body*, ***kwargs*)

**initializer**

**condition**

**iterator**

**body**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CIfElse**

Bases: *CStatement*

Represents an if-else construct in C.

**__init__**(*condition_and_nodes*, *else_node=None*, *simplify_else_scope=False*, *cstyle_ifs=True*, ***kwargs*)

**Parameters**
**condition_and_nodes** (*list[tuple[CExpression, CStatement | None]]*)

**condition_and_nodes**

**else_node**

**simplify_else_scope**

**cstyle_ifs**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CIfBreak**

Bases: *CStatement*

Represents an if-break statement in C.

**__init__**(*condition*, *cstyle_ifs=True*, ***kwargs*)

**condition**

**cstyle_ifs**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CBreak**

Bases: *CStatement*

Represents a break statement in C.

**__init__**(***kwargs*)

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CContinue**

Bases: *CStatement*

Represents a continue statement in C.

**__init__**(***kwargs*)

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CSwitchCase**

Bases: *CStatement*

Represents a switch-case statement in C.

**__init__**(*switch*, *cases*, *default*, ***kwargs*)

**switch**

**cases:** list[tuple[int | tuple[int], *CStatements*]]

**default**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CIncompleteSwitchCase**

Bases: *CStatement*

Represents an incomplete switch-case construct; this only appear in the decompilation output when switch-case structuring fails (for whatever reason).

**__init__**(*head*, *cases*, ***kwargs*)

**head**

**cases:** list[tuple[int, *CStatements*]]

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CAssignment**

Bases: *CStatement*

a = b

**__init__**(*lhs*, *rhs*, ***kwargs*)

**lhs**

**rhs**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CExpressionStatement**

Bases: *CStatement*

Wraps a CExpression so it can be used as a standalone statement.

expr;

__init__(*expr*, *returning=True*, ***kwargs*)

**Parameters**

- **expr** (CExpression)
- **returning** (*bool*)

**expr**

**returning**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CFunctionCall**

Bases: *CExpression*

func(arg0, arg1)

**Variables**
**callee_func** (Function) – The function getting called.

__init__(*callee_target*, *callee_func*, *args*, *show_demangled_name=True*, *show_disambiguated_name=True*, *tags=None*, ***, *codegen*, ***kwargs*)

**Parameters**
**show_disambiguated_name** (*bool*)

**callee_target**

**callee_func**: *Function* | None

**args**

**show_demangled_name**

**show_disambiguated_name**

**property prettify_thiscall**: bool

**property prototype**: *SimTypeFunction* | None

**property prototype_returnty**: *SimType*

Returns returnty and avoids creating the SimTypeFunction instance if the function prototype is not available. Instead of self.prototype.returnty, you should use self.prototype_returnty for better performance.

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CReturn**

Bases: *CStatement*

__init__(*retval*, ***kwargs*)

**retval**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CGoto**

Bases: *CStatement*

__init__(*target*, *target_idx*, ***kwargs*)

target: int | CExpression

target_idx

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CUnsupportedStatement

Bases: CStatement

A wrapper for unsupported AIL statement.

__init__(*stmt*, ***kwargs*)

stmt

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CDirtyStatement

Bases: CExpression

__init__(*dirty*, ***kwargs*)

**Parameters**
**dirty** (CDirtyExpression)

dirty

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CLabel

Bases: CStatement

Represents a label in C code.

__init__(*name*, ***kwargs*)

**Parameters**
**name** (str)

name

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CStructField

Bases: CExpression

__init__(*struct_type*, *offset*, *field*, ***kwargs*)

**Parameters**

- **struct_type** (SimStruct)
- **offset** (int)
- **field** (str)

struct_type

**offset**

**field**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.c.**CFakeVariable**

Bases: *CExpression*

An uninterpreted name to display in the decompilation output. Pretty much always represents an error?

**__init__**(*name*, *ty*, ***kwargs*)

**Parameters**

- **name** (*str*)
- **ty** (SimType)

**name**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.c.**CVariable**

Bases: *CExpression*

CVariable represents access to a variable with the specified type (*variable_type*).

*variable* must be a SimVariable.

**__init__**(*variable*, *unified_variable=None*, *variable_type=None*, *vvar_id=None*, ***kwargs*)

**Parameters**

**variable** (SimVariable)

**variable:** *SimVariable*

**unified_variable:** *SimVariable* | None

**variable_type:** *SimType* | None

**vvar_id**

**property type**

**property name**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.c.**CIndexedVariable**

Bases: *CExpression*

Represent a variable (an array) that is indexed.

**__init__**(*variable*, *index*, *variable_type=None*, ***kwargs*)

**Parameters**

- **variable** (CExpression)
- **index** (CExpression)

**index:** *CExpression*

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**collapsed**

**class** angr.analyses.decompiler.structured_codegen.c.**CVariableField**

Bases: *CExpression*

Represent a field of a variable.

**stl_accessor:** str | None = None

**__init__**(*variable*, *field*, *var_is_ptr=False*, ***kwargs*)

**Parameters**

- **variable** (CExpression)
- **field** (CStructField)
- **var_is_ptr** (*bool*)

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**collapsed**

**class** angr.analyses.decompiler.structured_codegen.c.**CUnaryOp**

Bases: *CExpression*

Unary operations.

**__init__**(*op*, *operand*, ***kwargs*)

**Parameters**

**operand** (CExpression)

**op**

**operand**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.c.**CBinaryOp**

Bases: *CExpression*

Binary operations.

**__init__**(*op*, *lhs*, *rhs*, ***kwargs*)

**op**

**lhs**

**rhs**

**common_type**

**static compute_common_type**(*op*, *lhs_ty*, *rhs_ty*)

**Return type**
*SimType*

**Parameters**

- **op** (*str*)
- **lhs_ty** (SimType)
- **rhs_ty** (SimType)

**property type**

**property op_precedence**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.c.**CTypeCast**

Bases: *CExpression*

**__init__**(*src_type*, *dst_type*, *expr*, ***kwargs*)

**Parameters**

- **src_type** (SimType | *None*)
- **dst_type** (SimType)
- **expr** (CExpression)

**src_type**

**dst_type**

**expr**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.c.**CConstant**

Bases: *CExpression*

**__init__**(*value*, *type_*, *reference_values=None*, ***kwargs*)

**Parameters**

- **type_** (SimType)
- **reference_values** (*dict*[SimType, *str* | bytes | *int* | *float* | Function | MemoryData] | *None*)

**value:** int | float | str

**reference_values**

**property fmt**

**property fmt_hex**

**property fmt_neg**

property fmt_char

property fmt_float

property fmt_double

property type

static str_to_c_str(*_str*, *prefix=''*, *maxlen=None*)

**Return type**
str

**Parameters**

- **prefix** (*str*)
- **maxlen** (*int | None*)

c_repr_chunks(*indent=0*, *asexpr=False*)

fmt_int(*value*)

Format an integer using the format setup of the current node.

**Parameters**
**value** (int) – The integer value to format.

**Return type**
str

**Returns**
The formatted string.

class angr.analyses.decompiler.structured_codegen.c.CRegister

Bases: *CExpression*

__init__(*reg*, ***kwargs*)

reg

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CITE

Bases: *CExpression*

__init__(*cond*, *iftrue*, *iffalse*, ***kwargs*)

cond

iftrue

iffalse

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression

Bases: *CExpression*

(stmt0, stmt1, stmt2, expr)

**__init__**(*stmts*, *expr*, ***kwargs*)

**Parameters**

- **stmts** (CStatements)
- **expr** (CExpression)

**stmts**

**expr**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CVEXCCallExpression**

Bases: *CExpression*

ccall_name(arg0, arg1, ...)

**__init__**(*callee*, *operands*, ***kwargs*)

**Parameters**

- **callee** (*str*)
- **operands** (*list[CExpression]*)

**callee**

**operands**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CVectorConvert**

Bases: *CExpression*

A lane-wise conversion (an AIL Convert with vector_count), rendered as an intrinsic-style call because C has no vector cast syntax.

**__init__**(*expr*, *operand*, ***kwargs*)

**Parameters**

- **expr** (Convert)
- **operand** (CExpression)

**expr**

**operand**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.**CDirtyExpression**

Bases: *CExpression*

Ideally all dirty expressions should be handled and converted to proper conversions during conversion from VEX to AIL. Eventually this class should not be used at all.

__init__(*dirty*, ***kwargs*)

dirty

property type

intrinsic_name()

Return the dirty callee if it is a clean C identifier, else None.

**Return type**
str | None

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.c.CClosingObject

Bases: object

A class to represent all objects that can be closed by it's correspodning character. Examples: (), {}, []

__init__(*opening_symbol*)

opening_symbol

class angr.analyses.decompiler.structured_codegen.c.CArrayTypeLength

Bases: object

A class to represent the type information of fixed-size array lengths. Examples: In "char foo[20]", this would be the "[20]".

__init__(*text*)

text

class angr.analyses.decompiler.structured_codegen.c.CStructFieldNameDef

Bases: object

A class to represent the name of a defined field in a struct. Needed because it's not a CVariable or a CStructField (because CStructField is the access of a CStructField). Example: In "struct foo { int bar; }, this would be "bar".

__init__(*name*)

name

class angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator

Bases: *BaseStructuredCodeGenerator*, *Analysis*, *Serializable*

__init__(*func*, *sequence*, *indent=0*, *cfg=None*, *func_args=None*, *binop_depth_cutoff=16*, *show_casts=True*, *braces_on_own_lines=True*, *use_compound_assignments=True*, *show_local_types=True*, *comment_gotos=False*, *cstyle_null_cmp=True*, *flavor=None*, *stmt_comments=None*, *expr_comments=None*, *show_externs=True*, *externs=None*, *const_formats=None*, *show_demangled_name=True*, *show_disambiguated_name=True*, *ail_graph=None*, *simplify_else_scope=True*, *cstyle_ifs=True*, *omit_func_header=False*, *display_block_addrs=False*, *display_vvar_ids=False*, *min_data_addr=4194304*, *notes=None*, *display_notes=True*, *max_str_len=None*, *prettify_thiscall=False*, *cstyle_void_param=True*, *indent_size=4*, *variable_map=None*, *stl_accessor_calls=False*)

**Parameters**

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)

- **min_data_addr** (*int*)
- **display_notes** (*bool*)
- **max_str_len** (*int | None*)
- **prettify_thiscall** (*bool*)
- **cstyle_void_param** (*bool*)
- **indent_size** (*int*)
- **variable_map** (VariableMap | *None*)
- **stl_accessor_calls** (*bool*)

**ailexpr2cnode:** dict[tuple[*Expression*, bool], *CExpression*] | None

**cnode2ailexpr:** dict[*CExpression*, *Expression*] | None

**map_ast_to_pos:** dict[*SimVariable*, set[*PositionMappingElement*]] | None

**map_addr_to_label:** dict[tuple[int, int | None], *CLabel*]

**cfunc:** *CFunction* | None

**cexterns:** set[*CVariable*] | None

**reapply_options**(*options*)

**cleanup**()

Remove existing rendering results.

**regenerate_text**()

Re-render text and re-generate all sorts of mapping information.

**Return type**
None

**render_text**(*cfunc*)

**Return type**
*RenderResult*

**Parameters**
**cfunc** (CFunction)

**render_notes**()

Render decompilation notes.

**Return type**
str

**Returns**
A string containing all notes.

**reload_variable_types**()

**Return type**
None

**default_simtype_from_bits**(*n*, *signed=True*)

**Return type**
*SimType*

**Parameters**

- **n** (*int*)
- **signed** (*bool*)

**variables_unify**(*v1*, *v2*)

**Return type**
bool

**Parameters**

- **v1** (VirtualVariable)
- **v2** (VirtualVariable)

**ERRNO_LOCATION_FUNCS = frozenset({'__errno', '__errno_location', '__error', '_errno'})**

The libc functions that return `&errno`. `errno` is a macro that dereferences one of them, so it never survives into a binary as a symbol; what reaches decompilation is `*(__errno_location())`, which is both correct and unreadable.

**class** angr.analyses.decompiler.structured_codegen.c.**CStructuredCodeWalker**

Bases: object

**handle**(*obj*)

**handle_default**(*obj*)

**handle_CFunction**(*obj*)

**handle_CStatements**(*obj*)

**handle_CWhileLoop**(*obj*)

**handle_CDoWhileLoop**(*obj*)

**handle_CForLoop**(*obj*)

**handle_CIfElse**(*obj*)

**handle_CIfBreak**(*obj*)

**handle_CSwitchCase**(*obj*)

**handle_CAssignment**(*obj*)

**handle_CExpressionStatement**(*obj*)

**handle_CFunctionCall**(*obj*)

**handle_CReturn**(*obj*)

**handle_CGoto**(*obj*)

**handle_CIndexedVariable**(*obj*)

**handle_CVariableField**(*obj*)

**handle_CUnaryOp**(*obj*)

**handle_CBinaryOp**(*obj*)

**handle_CTypeCast**(*obj*)

**handle_CITE**(*obj*)

**handle_CVectorConvert**(*obj*)

**handle_CVEXCCallExpression**(*obj*)

**class** angr.analyses.decompiler.structured_codegen.c.**MakeTypecastsImplicit**

Bases: *CStructuredCodeWalker*

**classmethod collapse**(*dst_ty*, *child*)

**Return type**
*CExpression*

**Parameters**

- **dst_ty** (SimType)
- **child** (CExpression)

**handle_CAssignment**(*obj*)

**handle_CFunctionCall**(*obj*)

**Parameters**
**obj** (CFunctionCall)

**handle_CReturn**(*obj*)

**Parameters**
**obj** (CReturn)

**handle_CBinaryOp**(*obj*)

**Parameters**
**obj** (CBinaryOp)

**handle_CTypeCast**(*obj*)

**Parameters**
**obj** (CTypeCast)

**class** angr.analyses.decompiler.structured_codegen.c.**FieldReferenceCleanup**

Bases: *CStructuredCodeWalker*

**handle_CTypeCast**(*obj*)

**class** angr.analyses.decompiler.structured_codegen.c.**PointerArithmeticFixer**

Bases: *CStructuredCodeWalker*

Before calling this fixer class, pointer arithmetics are purely integer-based and ignoring the pointer type.

For example, in the following case:

struct A* a_ptr; // assume struct A is 24 bytes in size a_ptr = a_ptr + 24;

It means adding 24 to the address of a_ptr, without considering the size of struct A. This fixer class will make pointer arithmetics aware of the pointer type. In this case, the fixer class will convert the code to a_ptr = a_ptr + 1.

**handle_CAssignment**(*obj*)

**Parameters**
**obj** (CAssignment)

**handle_CBinaryOp**(*obj*)

**Parameters**
**obj** (CBinaryOp)

### angr.analyses.decompiler.structured_codegen.c_serialize

Protobuf serialization helpers for the C AST defined in `c`.

The AST is serialized as a flat indexed table of `CConstructNode` cmessages: every `CConstruct` instance is assigned a non-zero `uint32` node_id at serialize time, and child CConstructs are referenced by id. This avoids unbounded recursion in protobuf encoding and lets `PositionMapping`, `cexterns`, and `map_addr_to_label` reference into the AST without duplicating subtrees.

**class** angr.analyses.decompiler.structured_codegen.c_serialize.**SerializeContext**

Bases: object

Tracks which nodes have been serialized while walking the C AST.

**__init__**()

**Return type**
None

**nodes:** list[*CConstructNode*]

**intern_type**(*t*)

Intern a SimType's JSON encoding and return its ref (index+1 into the type pool); 0 for None.

**Return type**
int

**Parameters**
**t** (SimType | *None*)

**property type_pool:** list[str]

**intern_tags**(*tags*)

Intern a sanitized tags dict and return its ref (index+1 into the tag pool); 0 for no tags.

**Return type**
int

**Parameters**
**tags** (*dict* | *None*)

**property tag_pool:** list[*CConstructTags*]

**intern_simvar**(*v*)

Intern a SimVariable's polymorphic payload and return its ref (index+1 into the pool); 0 for None.

**Return type**
int

**Parameters**
**v** (SimVariable | None)

**property simvar_pool: list[bytes]**

**add_cfuncall_config**(*node_id*, *show_demangled_name*, *show_disambiguated_name*)

**Return type**
None

**Parameters**

- **node_id** (*int*)
- **show_demangled_name** (*bool*)
- **show_disambiguated_name** (*bool*)

**property node_config: list[NodeConfigEntry]**

**serialize**(*node*)

Serialize node (recursively) and return its node_id (== node.idx). 0 indicates absent.

**Return type**
int

**Parameters**
**node** (CConstruct | None)

**class** angr.analyses.decompiler.structured_codegen.c_serialize.**ParseContext**

Bases: object

Tracks and resolves node.idx to the corresponding C AST.

**__init__**(*nodes_msg*, *project=None*, *kb=None*, *type_pool=()*, *tag_pool=()*, *simvar_pool=()*, *node_config=None*)

**Return type**
None

**project**

**kb**

**cfuncall_config**(*node_id*)

(show_demangled_name, show_disambiguated_name) for a CFunctionCall node; defaults to (True, True).

**Return type**
tuple[bool, bool]

**Parameters**
**node_id** (*int*)

**resolve_type**(*ref*)

Resolve a type-pool ref (index+1; 0 means absent) into a fresh SimType instance, arch-bound so it can report its size when re-rendered.

**Return type**
*SimType* | None

**Parameters**
**ref** (*int*)

**resolve_simvar**(*ref*)

Resolve a simvar-pool ref (index+1; 0 means absent) into a fresh SimVariable instance.

**Return type**
*SimVariable* | None

**Parameters**
**ref** (*int*)

**resolve_tags**(*ref*)

Resolve a tag-pool ref (index+1; 0 means no tags) into a fresh tags dict.

**Return type**
dict

**Parameters**
**ref** (*int*)

**resolve**(*node_id*)

**Parameters**
**node_id** (*int*)

**set_codegen**(*codegen*)

**Return type**
None

angr.analyses.decompiler.structured_codegen.c_serialize.**serialize_subtree**(*root*)

Serialize an AST subtree as a Codegen envelope carrying only the indexed table + root_id. For testing and any caller that wants to round-trip a CConstruct without a full codegen object.

**Return type**
bytes

**Parameters**
**root** (CConstruct)

angr.analyses.decompiler.structured_codegen.c_serialize.**parse_subtree**(*data*, *project=None*, *kb=None*)

**Return type**
*CConstruct* | None

**Parameters**
**data** (bytes)

angr.analyses.decompiler.structured_codegen.c_serialize.**serialize_codegen**(*codegen*)

Build a Codegen cmessage from a live CStructuredCodeGenerator instance.

**Return type**
*Codegen*

angr.analyses.decompiler.structured_codegen.c_serialize.**parse_codegen**(*msg*, ***, *project=None*, *kb=None*, *func=None*)

Create a CStructuredCodeGenerator from a Codegen cmessage. Bypasses __init__ (which runs the full decompilation pipeline) and populates the attributes directly. The parsed instance is suitable for display, navigation, and cache-validity checks; re-rendering and re-running analyses require `project` / `func` / `kb` to be reattached. Decompilation variables are read from `kb.dec_variables`.

angr.analyses.decompiler.structured_codegen.c_serialize.**register_all**()

Registers serializer/parser pairs for every concrete CConstruct subclass. Called from c.py at import time.

**Return type**
None

### angr.analyses.decompiler.structured_codegen.dummy

**class** angr.analyses.decompiler.structured_codegen.dummy.**DummyStructuredCodeGenerator**

Bases: *BaseStructuredCodeGenerator*, *Serializable*

A dummy structured code generator that only stores user-specified information.

**__init__**(*flavor*, *expr_comments=None*, *stmt_comments=None*, *configuration=None*, *const_formats=None*)

**Parameters**
**flavor** (*str*)

### angr.analyses.decompiler.structured_codegen.dwarf_import

**class** angr.analyses.decompiler.structured_codegen.dwarf_import.**ImportedLine**

Bases: object

**__init__**(*addr*)

**class** angr.analyses.decompiler.structured_codegen.dwarf_import.**ImportSourceCode**

Bases: *BaseStructuredCodeGenerator*, *Analysis*

**__init__**(*function*, *flavor='source'*, *source_root=None*, *encoding='utf-8'*)

**regenerate_text**()

### angr.analyses.decompiler.structured_codegen.rust

angr.analyses.decompiler.structured_codegen.rust.**unpack_typeref**(*ty*)

angr.analyses.decompiler.structured_codegen.rust.**unpack_pointer**(*ty*)

**Return type**
*SimType* | None

angr.analyses.decompiler.structured_codegen.rust.**unpack_array**(*ty*)

**Return type**
*SimType* | None

angr.analyses.decompiler.structured_codegen.rust.**squash_array_reference**(*ty*)

angr.analyses.decompiler.structured_codegen.rust.**qualifies_for_simple_cast**(*ty1*, *ty2*)

angr.analyses.decompiler.structured_codegen.rust.**qualifies_for_width_cast**(*ty*)

angr.analyses.decompiler.structured_codegen.rust.**qualifies_for_implicit_cast**(*ty1*, *ty2*)

angr.analyses.decompiler.structured_codegen.rust.**extract_terms**(*expr*)

**Return type**
tuple[int, list[tuple[int, *RustExpression*]]]

**Parameters**
**expr** (RustExpression)

angr.analyses.decompiler.structured_codegen.rust.**is_machine_word_size_type**(*type_*, *arch*)

**Return type**
bool

**Parameters**

- **type_** (SimType)
- **arch** (*Arch*)

angr.analyses.decompiler.structured_codegen.rust.**guess_value_type**(*value*, *project*)

**Return type**
*SimType* | None

**Parameters**

- **value** (*float*)
- **project** (Project)

angr.analyses.decompiler.structured_codegen.rust.**type_to_rust_repr_chunks**(*ty*, *name=None*, *name_type=None*, *full=False*, *indent_str=''*)

Helper generator function to turn a SimType into generated tuples of (C-string, AST node).

**Parameters**
**ty** (SimType)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustConstruct**

Bases: object

Represents a program construct in C. Acts as the base class for all other representation constructions.

**__init__**(*codegen*)

**codegen:** *RustStructuredCodeGenerator*

**c_repr**(*indent=0*, *pos_to_node=None*, *pos_to_addr=None*, *addr_to_pos=None*)

Creates the C representation of the code and displays it by constructing a large string. This function is called by each program function that needs to be decompiled. The map_pos_to_node and map_pos_to_addr act as position maps for the location of each variable and statement to be tracked for later GUI operations. The map_pos_to_addr also contains expressions that are nested inside of statements.

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**static indent_str**(*indent=0*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustFunction**

Bases: *RustConstruct*

Represents a function in Rust.

**__init__**(*addr*, *name*, *functy*, *arg_list*, *statements*, *variables_in_use*, *variable_manager*, *demangled_name=None*, *show_demangled_name=True*, ***kwargs*)

**Parameters**

- **functy** (RustSimTypeFunction)
- **arg_list** (*list[RustVariable]*)

**addr**

**name**

**functy**

**arg_list**

**statements**

**variables_in_use**

**variable_manager:** *VariableManagerInternal*

**demangled_name**

**unified_local_vars:** dict[*SimVariable*, set[tuple[*RustVariable*, *SimType*]]]

**show_demangled_name**

**get_unified_local_vars**()

**Return type**
dict[*SimVariable*, set[tuple[*RustVariable*, *SimType*]]]

**variable_list_repr_chunks**(*indent=0*)

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustStatement**

Bases: *RustConstruct*

Represents a statement in C.

**class** angr.analyses.decompiler.structured_codegen.rust.**RustExpression**

Bases: *RustConstruct*

Base class for C expressions.

**__init__**(*collapsed=False*, ***kwargs*)

**collapsed**

**property type**

**set_type**(*v*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustStatements**

Bases: *RustStatement*

Represents a sequence of statements in C.

**__init__**(*statements*, ***kwargs*)

**statements**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustAILBlock**

Bases: *RustStatement*

Represents a block of AIL statements.

**__init__**(*block*, ***kwargs*)

**block**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustLoop**

Bases: *RustStatement*

Represents a loop in Rust.

class angr.analyses.decompiler.structured_codegen.rust.**RustInfiniteLoop**

Bases: *RustLoop*

Represents an infinite loop in Rust.

**__init__**(*body*, *tags=None*, ***kwargs*)

**body**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustWhileLoop**

Bases: *RustLoop*

Represents a while loop in C.

**__init__**(*condition*, *body*, *tags=None*, ***kwargs*)

**condition**

**body**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustDoWhileLoop**

Bases: *RustLoop*

Represents a do-while loop in C.

**__init__**(*condition*, *body*, *tags=None*, ***kwargs*)

**condition**

**body**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustForLoop**

Bases: *RustStatement*

Represents a for-loop in C.

**__init__**(*initializer*, *condition*, *iterator*, *body*, *tags=None*, ***kwargs*)

**initializer**

**condition**

**iterator**

**body**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustIfElse**

Bases: *RustStatement*

Represents an if-else construct in Rust.

**__init__**(*condition_and_nodes*, *else_node=None*, *simplify_else_scope=False*, *cstyle_ifs=True*, *tags=None*, ***kwargs*)

**Parameters**

**condition_and_nodes** (*list[tuple[RustExpression, RustStatement | None]]*)

**condition_and_nodes**

**else_node**

**simplify_else_scope**

**cstyle_ifs**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustIfBreak**

Bases: *RustStatement*

Represents an if-break statement in C.

**__init__**(*condition*, *cstyle_ifs=True*, *tags=None*, ***kwargs*)

**condition**

**cstyle_ifs**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustBreak**

Bases: *RustStatement*

Represents a break statement in C.

__init__(*tags=None*, ***kwargs*)

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustContinue**

Bases: *RustStatement*

Represents a continue statement in C.

__init__(*tags=None*, ***kwargs*)

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustSwitchCase**

Bases: *RustStatement*

Represents a switch-case statement in C.

__init__(*switch*, *cases*, *default*, *tags=None*, ***kwargs*)

switch

cases: list[tuple[int | tuple[int], *RustStatements*]]

default

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustPatternMatch**

Bases: *RustStatement*

Represents a pattern-match statement in Rust.

__init__(*match_expr*, *cases*, *default*, *tags=None*, ***kwargs*)

scrutinee

arms: list[tuple[tuple[*EnumVariant*, tuple[*RustExpression*, ...]], *RustStatements*]]

default

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustIfLet**

Bases: *RustStatement*

Represents an if let statement in Rust.

__init__(*pattern*, *scrutinee*, *true_node*, *false_node*, *tags=None*, ***kwargs*)

pattern

scrutinee

**true_node**

**false_node**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustAssignment**

Bases: *RustStatement*

a = b

**__init__**(*lhs*, *rhs*, *tags=None*, ***kwargs*)

**lhs**

**rhs**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustFunctionLikeMacro**

Bases: *RustStatement*, *RustExpression*

**__init__**(*name*, *args*, *delimiter*, *is_expr*, *returnty=None*, *tags=None*, ***kwargs*)

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**property type**

**class** angr.analyses.decompiler.structured_codegen.rust.**RustFunctionCall**

Bases: *RustStatement*, *RustExpression*

func(arg0, arg1)

**Variables**

- **callee_func** (Function) – The function getting called.
- **is_expr** – True if the return value of the function is written to ret_expr; Essentially, ret_expr = call().

**__init__**(*callee_target*, *callee_func*, *args*, *returning=True*, *ret_expr=None*, *receiver=None*, *tags=None*, *is_expr=False*, *show_demangled_name=True*, *show_disambiguated_name=True*, *callsite_prototype=None*, ***kwargs*)

**Parameters**

- **is_expr** (*bool*)
- **show_disambiguated_name** (*bool*)
- **callsite_prototype** (RustSimTypeFunction | *None*)

**callee_target**

**callee_func: *Function* | None**

**args**

**returning**

**ret_expr**

**receiver**

**tags**

**is_expr**

**show_demangled_name**

**show_disambiguated_name**

**callsite_prototype**

**property prototype:** *SimTypeFunction* | None

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**Parameters**

- **indent** – Number of whitespace indentation characters.
- **asexpr** (bool) – True if this call is used as an expression (which means we will skip the generation of semicolons and newlines at the end of the call).

**class** angr.analyses.decompiler.structured_codegen.rust.**RustReturn**

Bases: *RustStatement*

**__init__**(*retval*, *tags=None*, ***kwargs*)

**retval**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustGoto**

Bases: *RustStatement*

**__init__**(*target*, *target_idx*, *tags=None*, ***kwargs*)

**target:** int | *RustExpression*

**target_idx**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustUnsupportedStatement**

Bases: *RustStatement*

A wrapper for unsupported AIL statement.

**__init__**(*stmt*, ***kwargs*)

**stmt**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustIncompleteSwitchCase**

Bases: *RustStatement*

An incomplete switch-case construct; only appears in the output when switch-case structuring failed.

**__init__**(*head*, *cases*, *tags=None*, ***kwargs*)

**head**

**cases:** list[tuple[int, *RustStatements*]]

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustDirtyStatement**

Bases: *RustStatement*

A dirty expression used as a statement, i.e. only for its side effects.

**__init__**(*dirty*, ***kwargs*)

**dirty**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustLabel**

Bases: *RustStatement*

Represents a label in C code.

**__init__**(*name*, *ins_addr*, *block_idx*, *tags=None*, ***kwargs*)

**Parameters**

- **name** (*str*)
- **ins_addr** (*int | None*)
- **block_idx** (*int | None*)

**name**

**ins_addr**

**block_idx**

**tags**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustStruct**

Bases: *RustExpression*

**__init__**(*name*, *fields*, *field_names*, *tags=None*, ***kwargs*)

**name**

**fields**

field_names

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustEnum**

Bases: *RustExpression*

__init__(*name*, *fields*, *tags=None*, ***kwargs*)

name

fields

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustStructField**

Bases: *RustExpression*

__init__(*struct_type*, *offset*, *field*, *tags=None*, ***kwargs*)

**Parameters**
**struct_type** (SimStruct)

struct_type

offset

field

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustArray**

Bases: *RustExpression*

__init__(*elements*, *tags=None*, ***kwargs*)

elements

tags

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustLet**

Bases: *RustExpression*

__init__(*let_expr*, *tags=None*, ***kwargs*)

**Parameters**
**let_expr** (Let)

let_expr

property variant

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

tags

class angr.analyses.decompiler.structured_codegen.rust.**RustStringLiteral**

Bases: *RustExpression*

__init__(*data*, *tags=None*, ***kwargs*)

data

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

tags

class angr.analyses.decompiler.structured_codegen.rust.**RustFakeVariable**

Bases: *RustExpression*

An uninterpreted name to display in the decompilation output. Pretty much always represents an error?

__init__(*name*, *ty*, *tags=None*, ***kwargs*)

**Parameters**

- **name** (*str*)
- **ty** (SimType)

name

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustVariable**

Bases: *RustExpression*

CVariable represents access to a variable with the specified type (*variable_type*).

*variable* must be a SimVariable.

__init__(*variable*, *unified_variable=None*, *variable_type=None*, *tags=None*, ***kwargs*)

**Parameters**
**variable** (SimVariable)

variable: *SimVariable*

unified_variable: *SimVariable* | None

variable_type: *SimType*

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustIndexedVariable**

Bases: *RustExpression*

Represent a variable (an array) that is indexed.

__init__(*variable*, *index*, *variable_type=None*, *tags=None*, ***kwargs*)

**Parameters**

- **variable** (RustExpression)
- **index** (RustExpression)

**index:** *RustExpression*

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

collapsed

class angr.analyses.decompiler.structured_codegen.rust.**RustVariableField**

Bases: *RustExpression*

Represent a field of a variable.

__init__(*variable*, *field*, *var_is_ptr=False*, *tags=None*, ***kwargs*)

**Parameters**

- **variable** (RustExpression)
- **field** (RustStructField)
- **var_is_ptr** (*bool*)

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

collapsed

class angr.analyses.decompiler.structured_codegen.rust.**RustUnaryOp**

Bases: *RustExpression*

Unary operations.

__init__(*op*, *operand*, *tags=None*, ***kwargs*)

**Parameters**
**operand** (RustExpression)

op

operand

tags

property type

c_repr_chunks(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustBinaryOp**

Bases: *RustExpression*

Binary operations.

**__init__**(*op*, *lhs*, *rhs*, *tags=None*, ***kwargs*)

**op**

**lhs**

**rhs**

**tags**

**common_type**

**static compute_common_type**(*op*, *lhs_ty*, *rhs_ty*)

**Return type**
*SimType*

**Parameters**

- **op** (*str*)
- **lhs_ty** (SimType)
- **rhs_ty** (SimType)

**property type**

**property op_precedence**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustTypeCast**

Bases: *RustExpression*

**__init__**(*src_type*, *dst_type*, *expr*, *tags=None*, ***kwargs*)

**Parameters**

- **src_type** (SimType | *None*)
- **dst_type** (SimType)
- **expr** (RustExpression)

**src_type**

**dst_type**

**expr**

**tags**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustConstant**

Bases: *RustExpression*

__init__(*value*, *type_*, *reference_values=None*, *tags=None*, ***kwargs*)

**Parameters**

- **type_** (SimType)
- **tags** (*dict | None*)

**value**

**reference_values**

**tags**

**property fmt**

**property fmt_hex**

**property fmt_neg**

**property fmt_char**

**property type**

**static str_to_rust_str**(*_str*, *is_heap_str=False*, *prefix=''*)

**Parameters**
**prefix** (str)

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**fmt_int**(*value*)

Format an integer using the format setup of the current node.

**Parameters**
**value** (int) – The integer value to format.

**Return type**
str

**Returns**
The formatted string.

class angr.analyses.decompiler.structured_codegen.rust.**RustRegister**

Bases: *RustExpression*

__init__(*reg*, *tags=None*, ***kwargs*)

**reg**

**tags**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

class angr.analyses.decompiler.structured_codegen.rust.**RustITE**

Bases: *RustExpression*

__init__(*cond*, *iftrue*, *iffalse*, *tags=None*, ***kwargs*)

**cond**

**iftrue**

**iffalse**

**tags**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustMultiStatementExpression**

Bases: *RustExpression*

(stmt0, stmt1, stmt2, expr)

**__init__**(*stmts*, *expr*, *tags=None*, ***kwargs*)

**Parameters**

- **stmts** (RustStatements)
- **expr** (RustExpression)

**stmts**

**expr**

**tags**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustVEXCCallExpression**

Bases: *RustExpression*

ccall_name(arg0, arg1, ...)

**__init__**(*callee*, *operands*, *tags=None*, ***kwargs*)

**Parameters**

- **callee** (*str*)
- **operands** (*list*[RustExpression])

**callee**

**operands**

**tags**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustVectorConvert**

Bases: *RustExpression*

A lane-wise conversion (an AIL Convert with vector_count), rendered as an intrinsic-style call.

__init__(*expr*, *operand*, ***kwargs*)

**Parameters**

- **expr** (Convert)
- **operand** (RustExpression)

**expr**

**operand**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustDirtyExpression**

Bases: *RustExpression*

Ideally all dirty expressions should be handled and converted to proper conversions during conversion from VEX to AIL. Eventually this class should not be used at all.

__init__(*dirty*, ***kwargs*)

**dirty**

**property type**

**c_repr_chunks**(*indent=0*, *asexpr=False*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustClosingObject**

Bases: object

A class to represent all objects that can be closed by it's correspodning character. Examples: (), {}, []

__init__(*opening_symbol*)

**opening_symbol**

**class** angr.analyses.decompiler.structured_codegen.rust.**RustArrayTypeLength**

Bases: object

A class to represent the type information of fixed-size array lengths. Examples: In "char foo[20]", this would be the "[20]".

__init__(*text*)

**text**

**class** angr.analyses.decompiler.structured_codegen.rust.**RustStructFieldNameDef**

Bases: object

A class to represent the name of a defined field in a struct. Needed because it's not a CVariable or a CStructField (because CStructField is the access of a CStructField). Example: In "struct foo { int bar; }, this would be "bar".

__init__(*name*)

**name**

**class** angr.analyses.decompiler.structured_codegen.rust.**RustStructuredCodeGenerator**

Bases: *BaseStructuredCodeGenerator*, *Analysis*

__init__(*func*, *sequence*, *indent=0*, *cfg=None*, *func_args=None*, *binop_depth_cutoff=16*, *show_casts=True*, *braces_on_own_lines=False*, *use_compound_assignments=True*, *show_local_types=False*, *comment_gotos=False*, *cstyle_null_cmp=True*, *flavor=None*, *stmt_comments=None*, *expr_comments=None*, *show_externs=False*, *externs=None*, *const_formats=None*, *show_demangled_name=True*, *show_disambiguated_name=True*, *ail_graph=None*, *simplify_else_scope=True*, *cstyle_ifs=True*, *omit_func_header=False*, *display_block_addrs=False*, *display_vvar_ids=False*, *min_data_addr=4194304*, *notes=None*, *display_notes=True*, *max_str_len=None*, *prettify_thiscall=False*, *cstyle_void_param=True*, *indent_size=4*, *variable_map=None*, *stl_accessor_calls=False*)

**Parameters**

- **func_args** (*list[SimVariable] | None*)
- **binop_depth_cutoff** (*int*)
- **min_data_addr** (*int*)
- **display_notes** (*bool*)
- **max_str_len** (*int | None*)
- **prettify_thiscall** (*bool*)
- **cstyle_void_param** (*bool*)
- **indent_size** (*int*)
- **variable_map** (*VariableMap | None*)
- **stl_accessor_calls** (*bool*)

**ailexpr2cnode:** dict[tuple[*Expression*, bool], *RustExpression*] | None

**cnode2ailexpr:** dict[*RustExpression*, *Expression*] | None

**expr_comments:** dict[int, str]

**stmt_comments:** dict[int, str]

**const_formats:** dict[Any, dict[str, Any]]

**map_ast_to_pos:** dict[*SimVariable*, set[*PositionMappingElement*]] | None

**map_addr_to_label:** dict[tuple[int | None, int | None], *RustLabel*]

**rust_func:** *RustFunction* | None

**cexterns:** set[*RustVariable*] | None

**reapply_options**(*options*)

**cleanup**()

Remove existing rendering results.

**regenerate_text**()

Re-render text and re-generate all sorts of mapping information.

**Return type**

None

**RENDER_TYPE**

alias of tuple[str, *PositionMapping*, *PositionMapping*, *InstructionMapping*, dict[Any, set[Any]]]

**render_text**(*rust_func*)

**Return type**
RENDER_TYPE

**Parameters**
**rust_func** (RustFunction)

**render_notes**()

Render decompilation notes.

**Return type**
str

**Returns**
A string containing all notes.

**reload_variable_types**()

**Return type**
None

**default_simtype_from_size**(*n*, *signed=True*)

**Return type**
*RustSimTypeInt*

**Parameters**

- **n** (*int*)
- **signed** (*bool*)

**class** angr.analyses.decompiler.structured_codegen.rust.**RustStructuredCodeWalker**

Bases: object

**classmethod handle**(*obj*)

**classmethod handle_default**(*obj*)

**classmethod handle_RustFunction**(*obj*)

**classmethod handle_RustStatements**(*obj*)

**classmethod handle_RustWhileLoop**(*obj*)

**classmethod handle_RustInfiniteLoop**(*obj*)

**classmethod handle_RustDoWhileLoop**(*obj*)

**classmethod handle_RustForLoop**(*obj*)

**classmethod handle_RustIfElse**(*obj*)

**classmethod handle_RustIfBreak**(*obj*)

**classmethod handle_RustSwitchCase**(*obj*)

classmethod **handle_RustAssignment**(*obj*)

classmethod **handle_RustFunctionCall**(*obj*)

classmethod **handle_RustReturn**(*obj*)

classmethod **handle_RustGoto**(*obj*)

classmethod **handle_RustIndexedVariable**(*obj*)

classmethod **handle_RustVariableField**(*obj*)

classmethod **handle_RustUnaryOp**(*obj*)

classmethod **handle_RustBinaryOp**(*obj*)

classmethod **handle_RustTypeCast**(*obj*)

classmethod **handle_RustITE**(*obj*)

classmethod **handle_RustVectorConvert**(*obj*)

classmethod **handle_RustVEXCCallExpression**(*obj*)

class angr.analyses.decompiler.structured_codegen.rust.**MakeTypecastsImplicit**

Bases: *RustStructuredCodeWalker*

classmethod **collapse**(*dst_ty*, *child*)

**Return type**
*RustExpression*

**Parameters**

- **dst_ty** (SimType | *None*)
- **child** (RustExpression)

classmethod **handle_RustAssignment**(*obj*)

classmethod **handle_RustFunctionCall**(*obj*)

**Parameters**
**obj** (RustFunctionCall)

classmethod **handle_RustReturn**(*obj*)

**Parameters**
**obj** (RustReturn)

classmethod **handle_RustBinaryOp**(*obj*)

**Parameters**
**obj** (RustBinaryOp)

classmethod **handle_RustTypeCast**(*obj*)

**Parameters**
**obj** (RustTypeCast)

class angr.analyses.decompiler.structured_codegen.rust.**FieldReferenceCleanup**

Bases: *RustStructuredCodeWalker*

classmethod handle_RustTypeCast(*obj*)

class angr.analyses.decompiler.structured_codegen.rust.PointerArithmeticFixer

Bases: *RustStructuredCodeWalker*

Before calling this fixer class, pointer arithmetics are purely integer-based and ignoring the pointer type.

For example, in the following case:

struct A* a_ptr; // assume struct A is 24 bytes in size a_ptr = a_ptr + 24;

It means adding 24 to the address of a_ptr, without considering the size of struct A. This fixer class will make pointer arithmetics aware of the pointer type. In this case, the fixer class will convert the code to a_ptr = a_ptr + 1.

classmethod handle_RustBinaryOp(*obj*)

**Parameters**
**obj** (RustBinaryOp)

### angr.analyses.decompiler.structurer_nodes

exception angr.analyses.decompiler.structurer_nodes.EmptyBlockNotice

Bases: Exception

class angr.analyses.decompiler.structurer_nodes.MultiNode

Bases: object

__init__(*nodes*, *addr=None*, *idx=None*)

**Parameters**

- **nodes** (*Iterable[Block]*)
- **addr** (*int | None*)
- **idx** (*int | None*)

nodes: list[Block]

addr

idx

copy()

**Return type**
*MultiNode*

dbg_repr(*indent=0*)

**Return type**
str

class angr.analyses.decompiler.structurer_nodes.BaseNode

Bases: object

__init__(*addr*)

**Parameters**
**addr** (*int | None*)

addr

static test_empty_node(*node*)

static test_empty_condition_node(*cond_node*)

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structurer_nodes.**SequenceNode**

Bases: BaseNode

__init__(*addr*, *nodes=None*)

**Parameters**
**addr** (*int | None*)

nodes

add_node(*node*)

insert_node(*pos*, *node*)

remove_node(*node*)

node_position(*node*)

copy()

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structurer_nodes.**CodeNode**

Bases: BaseNode

__init__(*node*, *reaching_condition*, *addr=None*, *idx=None*)

**Parameters**

- **addr** (*int | None*)
- **idx** (*int | None*)

node

reaching_condition

idx

dbg_repr(*indent=0*)

copy()

class angr.analyses.decompiler.structurer_nodes.**ConditionNode**

Bases: BaseNode

__init__(*addr*, *reaching_condition*, *condition*, *true_node*, *false_node=None*)

reaching_condition

condition

true_node

**false_node**

**dbg_repr**(*indent=0*)

**node**

class angr.analyses.decompiler.structurer_nodes.**CascadingConditionNode**

Bases: *BaseNode*

**__init__**(*addr*, *condition_and_nodes*, *else_node=None*)

**Parameters**

- **condition_and_nodes** (*list[tuple[Any, BaseNode | Block | MultiNode]]*)
- **else_node** (*BaseNode | None*)

**condition_and_nodes**

**else_node**

**dbg_repr**(*indent=0*)

class angr.analyses.decompiler.structurer_nodes.**LoopNode**

Bases: *BaseNode*

**__init__**(*sort*, *condition*, *sequence_node*, *addr=None*, *continue_addr=None*, *initializer=None*, *iterator=None*)

**Parameters**

- **sort** (*str*)
- **condition** (*Expression | None*)
- **sequence_node** (*SequenceNode*)
- **addr** (*int | None*)
- **continue_addr** (*int | None*)
- **initializer** (*Assignment | None*)
- **iterator** (*Assignment | None*)

**sort**: *str*

**condition**: *Expression* | None

**sequence_node**: *SequenceNode*

**initializer**: *Assignment* | None

**iterator**: *Assignment* | None

**copy**()

**property continue_addr**

**dbg_repr**(*indent=0*)

class angr.analyses.decompiler.structurer_nodes.**BreakNode**

Bases: *BaseNode*

__init__(*addr*, *target*)

target

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structurer_nodes.ContinueNode

Bases: *BaseNode*

__init__(*addr*, *target*)

target

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structurer_nodes.ConditionalBreakNode

Bases: *BreakNode*

__init__(*addr*, *condition*, *target*)

condition

dbg_repr(*indent=0*)

class angr.analyses.decompiler.structurer_nodes.SwitchCaseNode

Bases: *BaseNode*

__init__(*switch_expr*, *cases*, *default_node*, *addr=None*)

**Parameters**
**cases** (*OrderedDict[int | tuple[int, ...], SequenceNode]*)

switch_expr

cases: OrderedDict[int | tuple[int, ...], *SequenceNode*]

default_node

dbg_repr(*indent=0*)

**Return type**
str

class angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseNode

Bases: *BaseNode*

Describes an incomplete set of switch-case nodes. Usually an intermediate result. Should always be restructured into a SwitchCaseNode by the end of structuring. Only used in Phoenix structurer.

__init__(*addr*, *head*, *cases*)

**Parameters**
**cases** (*list*)

head

cases: list

class angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseHeadStatement

Bases: *Statement*

Describes a switch-case head. This is only created by LoweredSwitchSimplifier.

kind = 'IncompleteSwitchCaseHead'

pykind = 'IncompleteSwitchCaseHead'

__init__(*idx*, *switch_variable*, *case_addrs*, *peephole_optimized=False*, ***kwargs*)

**Parameters**
**peephole_optimized** (*bool*)

switch_variable

case_addrs: list[tuple[Block, int | str, int, int | None, int]]

peephole_optimized

replace(*old_expr*, *new_expr*)

property depth: int

addr

likes(*other*)

matches(*other*)

### angr.analyses.decompiler.structuring

angr.analyses.decompiler.structuring.**DEFAULT_STRUCTURER**

alias of *SAILRStructurer*

class angr.analyses.decompiler.structuring.**DreamStructurer**

Bases: *StructurerBase*

Structure a region using a structuring algorithm that is similar to the one in Dream decompiler (described in the "no more gotos" paper). Note that this implementation has quite a few improvements over the original described version and *should not* be used to evaluate the performance of the original algorithm described in that paper.

The current function graph is provided so that we can detect certain edge cases, for example, jump table entries no longer exist due to empty node removal during structuring or prior steps.

NAME = 'dream'

SUPPORTS_OVERLAYS = True

__init__(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, ***kwargs*)

**Parameters**

- **func** (Function | *None*)
- **case_entry_to_switch_head** (*dict[int, int] | None*)

class angr.analyses.decompiler.structuring.**PhoenixStructurer**

Bases: *StructurerBase*

Structure a region using a structuring algorithm that is similar to the one in Phoenix decompiler (described in the "phoenix decompiler" paper). Note that this implementation has quite a few improvements over the original described version and *should not* be used to evaluate the performance of the original algorithm described in that paper.

NAME = 'phoenix'

SUPPORTS_OVERLAYS = True

__init__(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, *improve_algorithm=False*, *use_multistmtexprs=MultiStmtExprMode.MAX_ONE_CALL*, *multistmtexpr_stmt_threshold=5*, ***kwargs*)

**Parameters**

- **func** (Function | *None*)
- **case_entry_to_switch_head** (*dict[int, int] | None*)
- **use_multistmtexprs** (MultiStmtExprMode)
- **multistmtexpr_stmt_threshold** (*int*)

replace_nodes_both(*old_node_0*, *new_node*, *old_node_1=None*, *self_loop=True*, *drop_refinement_marks=False*)

Replace one or two nodes with a new node in the region: the member view, the with-successors view, and the complete graph are all updated by the single overlay operation. If `old_node_1` is a successor of the region rather than a member, it is absorbed into the new node in this region's views only.

**Parameters**

**drop_refinement_marks** (*bool*)

static dump_graph(*graph*, *path*)

**Return type**

None

**Parameters**

- **graph** (*DiGraph*)
- **path** (*str*)

static switch_case_entry_node_has_common_successor_case_1(*graph*, *jump_table*, *case_nodes*, *node_pred*)

**Return type**

bool

static switch_case_entry_node_has_common_successor_case_2(*graph*, *jump_table*, *case_nodes*, *node_pred*)

**Return type**

bool

class angr.analyses.decompiler.structuring.RecursiveStructurer

Bases: *Analysis*

Recursively structure a region and all of its subregions.

__init__(*region*, *cond_proc=None*, *func=None*, *structurer_cls=None*, ***, *ail_manager*, ***kwargs*)

**Parameters**

- **func** (Function | *None*)
- **structurer_cls** (type | *None*)

- **ail_manager** (*Manager*)

**class** angr.analyses.decompiler.structuring.**SAILRStructurer**

Bases: *PhoenixStructurer*

The SAILR structuring algorithm is the phoenix-based algorithm from the USENIX 2024 paper SAILR. The entirety of the algorithm is implemented across this class and various optimization passes in the decompiler. To find each optimization class, simply search for optimizations which reference this class.NAME.

At a high-level, SAILR does three things different from the traditional Phoenix schema-based algorithm:

1. It recursively structures the graph, rather than doing it in a single pass. This allows decisions to be made based on the current state of what the decompilation would look like.
2. It performs deoptimizations targeting specific optimizations that introduces gotos and mis-structured code. It can only do this because of the recursive nature of the algorithm.
3. It uses a more advanced heuristic for virtualizing edges, which is implemented in this class.

Additionally, some changes in Phoenix are only activated when SAILR is used.

**NAME = 'sailr'**

**__init__**(*region*, *improve_phoenix=True*, *postdom_max_edges=10*, *postdom_max_graph_size=50*, ***kwargs*)

angr.analyses.decompiler.structuring.**structurer_class_from_name**(*name*)

**Return type**
type | None

**Parameters**
**name** (*str*)

## Submodules

| |
|---|
| *dream* |
| *phoenix* |
| *recursive_structurer* |
| *sailr* |
| *structurer_base* |

### angr.analyses.decompiler.structuring.dream

**class** angr.analyses.decompiler.structuring.dream.**DreamStructurer**

Bases: *StructurerBase*

Structure a region using a structuring algorithm that is similar to the one in Dream decompiler (described in the "no more gotos" paper). Note that this implementation has quite a few improvements over the original described version and *should not* be used to evaluate the performance of the original algorithm described in that paper.

The current function graph is provided so that we can detect certain edge cases, for example, jump table entries no longer exist due to empty node removal during structuring or prior steps.

**NAME: str = 'dream'**

**SUPPORTS_OVERLAYS = True**

__init__(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, ***kwargs*)

**Parameters**

- **func** (Function | *None*)
- **case_entry_to_switch_head** (*dict[int, int] | None*)

### angr.analyses.decompiler.structuring.phoenix

**exception** angr.analyses.decompiler.structuring.phoenix.**GraphChangedNotification**

Bases: Exception

A notification for graph that is currently worked on being changed. Once this notification is caught, the graph schema matching process for the current region restarts.

**class** angr.analyses.decompiler.structuring.phoenix.**MultiStmtExprMode**

Bases: StrEnum

Mode of multi-statement expression creation during structuring.

**NEVER = 'Never'**

**ALWAYS = 'Always'**

**MAX_ONE_CALL = 'Only when less than one call'**

__new__(*value*)

**class** angr.analyses.decompiler.structuring.phoenix.**PhoenixStructurer**

Bases: *StructurerBase*

Structure a region using a structuring algorithm that is similar to the one in Phoenix decompiler (described in the "phoenix decompiler" paper). Note that this implementation has quite a few improvements over the original described version and *should not* be used to evaluate the performance of the original algorithm described in that paper.

**NAME: str = 'phoenix'**

**SUPPORTS_OVERLAYS = True**

__init__(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, *improve_algorithm=False*, *use_multistmtexprs=MultiStmtExprMode.MAX_ONE_CALL*, *multistmtexpr_stmt_threshold=5*, ***kwargs*)

**Parameters**

- **func** (Function | *None*)
- **case_entry_to_switch_head** (*dict[int, int] | None*)
- **use_multistmtexprs** (MultiStmtExprMode)
- **multistmtexpr_stmt_threshold** (*int*)

**whitelist_edges: set[tuple[int, int]]**

**switch_case_known_heads: set[Block | *BaseNode*]**

**dowhile_known_tail_nodes:** set

**replace_nodes_both**(*old_node_0*, *new_node*, *old_node_1=None*, *self_loop=True*, *drop_refinement_marks=False*)

Replace one or two nodes with a new node in the region: the member view, the with-successors view, and the complete graph are all updated by the single overlay operation. If `old_node_1` is a successor of the region rather than a member, it is absorbed into the new node in this region's views only.

**Parameters**
**drop_refinement_marks** (*bool*)

**static dump_graph**(*graph*, *path*)

**Return type**
None

**Parameters**

- **graph** (*DiGraph*)
- **path** (*str*)

**static switch_case_entry_node_has_common_successor_case_1**(*graph*, *jump_table*, *case_nodes*, *node_pred*)

**Return type**
bool

**static switch_case_entry_node_has_common_successor_case_2**(*graph*, *jump_table*, *case_nodes*, *node_pred*)

**Return type**
bool

### angr.analyses.decompiler.structuring.recursive_structurer

**class** angr.analyses.decompiler.structuring.recursive_structurer.**RecursiveStructurer**

Bases: *Analysis*

Recursively structure a region and all of its subregions.

**__init__**(*region*, *cond_proc=None*, *func=None*, *structurer_cls=None*, ***, *ail_manager*, ***kwargs*)

**Parameters**

- **func** (Function | *None*)
- **structurer_cls** (type | *None*)
- **ail_manager** (*Manager*)

### angr.analyses.decompiler.structuring.sailr

**class** angr.analyses.decompiler.structuring.sailr.**SAILRStructurer**

Bases: *PhoenixStructurer*

The SAILR structuring algorithm is the phoenix-based algorithm from the USENIX 2024 paper SAILR. The entirety of the algorithm is implemented across this class and various optimization passes in the decompiler. To find each optimization class, simply search for optimizations which reference this class.NAME.

At a high-level, SAILR does three things different from the traditional Phoenix schema-based algorithm:

1. It recursively structures the graph, rather than doing it in a single pass. This allows decisions to be made based on the current state of what the decompilation would look like.
2. It performs deoptimizations targeting specific optimizations that introduces gotos and mis-structured code. It can only do this because of the recursive nature of the algorithm.
3. It uses a more advanced heuristic for virtualizing edges, which is implemented in this class.

Additionally, some changes in Phoenix are only activated when SAILR is used.

**NAME: str = 'sailr'**

**__init__**(*region*, *improve_phoenix=True*, *postdom_max_edges=10*, *postdom_max_graph_size=50*, ***kwargs*)

### angr.analyses.decompiler.structuring.structurer_base

**class** angr.analyses.decompiler.structuring.structurer_base.**StructurerBase**

Bases: *Analysis*

The base class for analysis passes that structures a region.

The current function graph is provided so that we can detect certain edge cases, for example, jump table entries no longer exist due to empty node removal during structuring or prior steps.

**NAME: str = 'StructurerBase'**

**__init__**(*region*, *parent_map=None*, *condition_processor=None*, *func=None*, *case_entry_to_switch_head=None*, *parent_region=None*, *jump_tables=None*, *ail_manager=None*, ***kwargs*)

**Parameters**

- **func** (*Function | None*)
- **case_entry_to_switch_head** (*dict[int, int] | None*)
- **jump_tables** (*dict[int, IndirectJump] | None*)
- **ail_manager** (*Manager | None*)

**static remove_redundant_jumps**(*seq*, *ail_manager*)

Remove all redundant jumps.

**Parameters**

- **seq** (SequenceNode) – The SequenceNode instance to handle.
- **ail_manager** (Manager) – The AIL manager to create new atoms if needed.

**Returns**

A processed SequenceNode.

**replace_nodes**(*graph*, *old_node_0*, *new_node*, *old_node_1=None*, *self_loop=True*)

**static replace_node_in_node**(*parent_node*, *old_node*, *new_node*)

**Return type**

None

**Parameters**

- **parent_node** (BaseNode)

- **old_node** (BaseNode | *Block* | MultiNode)
- **new_node** (BaseNode | *Block* | MultiNode)

static **is_a_jump_target**(*stmt*, *addr*)

**Return type**
bool

**Parameters**

- **stmt** (ConditionalJump | Jump | Statement)
- **addr** (*int*)

static **has_nonlabel_nonphi_statements**(*node*)

**Return type**
bool

**Parameters**
**node** (BaseNode)

### angr.analyses.decompiler.utils

angr.analyses.decompiler.utils.**remove_last_statement**(*node*)

angr.analyses.decompiler.utils.**remove_last_statements**(*node*)

**Return type**
bool

angr.analyses.decompiler.utils.**append_statement**(*node*, *stmt*)

angr.analyses.decompiler.utils.**replace_last_statement**(*node*, *old_stmt*, *new_stmt*)

angr.analyses.decompiler.utils.**extract_jump_targets**(*stmt*)

Extract concrete goto targets from a Jump or a ConditionalJump statement.

**Parameters**
**stmt** – The statement to analyze.

**Returns**
A list of known concrete jump targets.

**Return type**
list

angr.analyses.decompiler.utils.**switch_extract_cmp_bounds**(*last_stmt*)

Check the last statement of the switch-case header node, and extract lower+upper bounds for the comparison.

**Parameters**
**last_stmt** (*ConditionalJump* | *Statement*) – The last statement of the switch-case header node.

**Return type**
tuple[Any, int, int] | None

**Returns**
A tuple of (comparison expression, lower bound, upper bound), or None

angr.analyses.decompiler.utils.**switch_extract_cmp_bounds_from_condition**(*cond*)

**Return type**
tuple[Any, int, int] | None

**Parameters**
**cond** (Expression)

angr.analyses.decompiler.utils.**switch_extract_switch_expr_from_jump_target**(*target*)

Extract the switch expression from the indirect jump target expression.

**Parameters**
**target** (*Expression*) – The target of the indirect jump statement.

**Return type**
*Expression* | None

**Returns**
The extracted expression if successful, or None otherwise.

angr.analyses.decompiler.utils.**switch_extract_bitwiseand_jumptable_info**(*last_stmt*)

Check the last statement of the switch-case header node (whose address is loaded from a jump table and computed using an index) and extract necessary information for rebuilding the switch-case construct.

An example of the statement:

```
Goto(Conv(32->s64, (
    Load(addr=(0x4530e4<64> + (Conv(32->64, (Conv(64->32, vvar_287{reg 32}) & 0x3
↪<32>)) * 0x4<64>)),
         size=4, endness=Iend_LE) + 0x4530e4<32>))
)
```

Another example:

```
Load(addr=(((vvar_9{reg 36} & 0x3<32>) * 0x4<32>) + 0x42cd28<32>), size=4,␣
↪endness=Iend_LE)
```

**Parameters**
**last_stmt** (*Jump*) – The last statement of the switch-case header node.

**Return type**
tuple[Any, int, int] | None

**Returns**
A tuple of `(index expression, lower bound, upper bound)`, or None.

angr.analyses.decompiler.utils.**get_ast_subexprs**(*claripy_ast*)

angr.analyses.decompiler.utils.**insert_node**(*parent*, *insert_location*, *node*, *node_idx*, *label=None*)

**Parameters**

- **insert_location** (*str*)
- **node_idx** (*int*)

angr.analyses.decompiler.utils.**to_ail_supergraph**(*transition_graph*, *allow_fake=False*)

Takes an AIL graph and converts it into a AIL graph that treats calls and redundant jumps as parts of a bigger block instead of transitions. Calls to returning functions do not terminate basic blocks.

Based on region_identifier super_graph

**Return type**
DiGraph

**Returns**
A converted super transition graph

**Parameters**
**transition_graph** (*DiGraph*)

angr.analyses.decompiler.utils.**is_empty_node**(*node*)

**Return type**
bool

angr.analyses.decompiler.utils.**is_empty_or_label_only_node**(*node*)

**Return type**
bool

angr.analyses.decompiler.utils.**has_nonlabel_statements**(*block*)

**Return type**
bool

**Parameters**
**block** (*Block*)

angr.analyses.decompiler.utils.**has_nonlabel_nonphi_statements**(*block*)

**Return type**
bool

**Parameters**
**block** (*Block*)

angr.analyses.decompiler.utils.**first_nonlabel_statement**(*block*)

**Return type**
*Statement* | None

**Parameters**
**block** (*Block* | MultiNode)

angr.analyses.decompiler.utils.**first_nonlabel_statement_id**(*block*)

**Return type**
int | None

**Parameters**
**block** (*Block*)

angr.analyses.decompiler.utils.**first_nonlabel_nonphi_statement**(*block*)

**Return type**
*Statement* | None

**Parameters**
**block** (*Block* | MultiNode)

angr.analyses.decompiler.utils.**last_nonlabel_statement**(*block*)

**Return type**
*Statement* | None

**Parameters**
**block** (*Block*)

angr.analyses.decompiler.utils.**last_node**(*node*)

Get the last node in a sequence or code node.

**Return type**
*BaseNode* | Block | None

**Parameters**
**node** (BaseNode)

angr.analyses.decompiler.utils.**first_nonlabel_node**(*seq*)

**Return type**
*BaseNode* | Block | None

**Parameters**
**seq** (SequenceNode)

angr.analyses.decompiler.utils.**first_nonlabel_nonphi_node**(*seq*)

**Return type**
*BaseNode* | Block | None

**Parameters**
**seq** (SequenceNode)

angr.analyses.decompiler.utils.**remove_labels**(*graph*)

**Parameters**
**graph** (*DiGraph*)

angr.analyses.decompiler.utils.**add_labels**(*graph*, *ail_manager*)

**Parameters**

- **graph** (*DiGraph*)
- **ail_manager** (*Manager*)

angr.analyses.decompiler.utils.**update_labels**(*graph*, *ail_manager*)

A utility function to recreate the labels for every node in an AIL graph. This useful when you are working with a graph where only _some_ of the nodes have labels.

**Parameters**

- **graph** (*DiGraph*)
- **ail_manager** (*Manager*)

angr.analyses.decompiler.utils.**structured_node_has_multi_predecessors**(*node*, *graph*)

**Return type**
bool

**Parameters**

- **node** (SequenceNode | MultiNode | *Block*)
- **graph** (*DiGraph*)

angr.analyses.decompiler.utils.**structured_node_is_simple_return**(*node*, *graph*, *use_packed_successors=False*)

Check if a "simple return" is contained within the node. A simple return looks like this:

```
if (cond) {
  // simple return
  ...
  return 0;
}
...
```

Returns true on any block ending in linear statements and a return.

**Return type**
bool

**Parameters**

- **node** (SequenceNode | MultiNode)
- **graph** (*DiGraph*)

angr.analyses.decompiler.utils.**structured_node_is_simple_return_strict**(*node*)

Returns True iff the node exclusively contains a return statement.

**Return type**
bool

**Parameters**
**node** (BaseNode | SequenceNode | MultiNode | *Block*)

angr.analyses.decompiler.utils.**is_statement_terminating**(*stmt*, *functions*)

**Return type**
bool

**Parameters**
**stmt** (Statement)

angr.analyses.decompiler.utils.**peephole_optimize_exprs**(*block*, *expr_opts*, *walker=None*)

angr.analyses.decompiler.utils.**peephole_optimize_expr**(*expr*, *expr_opts*)

**Parameters**

- **expr** (Expression)
- **expr_opts** (*list*[PeepholeOptimizationExprBase])

angr.analyses.decompiler.utils.**copy_graph**(*graph*)

Copy AIL Graph.

**Returns**
A copy of the AIl graph.

angr.analyses.decompiler.utils.**build_stmt_opts_by_kind**(*stmt_opts*)

Index statement-peephole optimizers by `kind` string.

angr.analyses.decompiler.utils.**peephole_optimize_stmts**(*block*, *stmt_opts*, *, *stmt_opts_by_kind=None*, *fixpoint_exprs=None*)

**Parameters**
**fixpoint_exprs** – IDs of the statements the preceding expression pass left untouched *and* whose expression optimizers all reported `fixpoint_reached`.

angr.analyses.decompiler.utils.**match_stmt_classes**(*all_stmts*, *idx*, *stmt_class_seq*)

**Return type**
bool

**Parameters**

- **all_stmts** (*list*)
- **idx** (*int*)
- **stmt_class_seq** (*Iterable[type]*)

angr.analyses.decompiler.utils.**peephole_optimize_multistmts**(*block*, *stmt_opts*)

**Parameters**
**stmt_opts** (*list[PeepholeOptimizationMultiStmtBase]*)

angr.analyses.decompiler.utils.**decompile_functions**(*path*, *functions=None*, *structurer=None*, *catch_errors=False*, *show_casts=True*, *base_address=None*, *preset=None*, *cca=False*, *cca_callsites=True*, *llm=False*, *progressbar=False*, *postmortem=False*)

Decompile a binary into a set of functions.

**Parameters**

- **path** – The path to the binary to decompile.
- **functions** (list[int | str] | None) – The functions to decompile. If None, all functions will be decompiled.
- **structurer** (str | None) – The structuring algorithms to use.
- **catch_errors** (bool) – The structuring algorithms to use.
- **show_casts** (bool) – Whether to show casts in the decompiled output.
- **base_address** (int | None) – The base address of the binary.
- **preset** (str | None) – The configuration preset to use during decompilation.
- **cca** (*bool*)
- **cca_callsites** (*bool*)
- **llm** (*bool*)
- **progressbar** (*bool*)
- **postmortem** (*bool*)

**Return type**
str

**Returns**
The decompilation of all functions appended in order.

angr.analyses.decompiler.utils.**calls_in_graph**(*graph*, *consider_conditions=False*)

Counts the number of calls in a graph full of AIL Blocks

**Return type**
int

**Parameters**

- **graph** (*DiGraph*)
- **consider_conditions** (*bool*)

angr.analyses.decompiler.utils.**call_exprs_in_graph**(*graph*, *consider_conditions=False*)

Return a list of all call expressions in a given AIL graph.

**Return type**
list[tuple[tuple[tuple[int, int | None], int], *Call*]]

**Parameters**

- **graph** (*DiGraph*)
- **consider_conditions** (*bool*)

angr.analyses.decompiler.utils.**has_addr_dups**(*graph*)

**Parameters**
**graph** (*networkx.DiGraph[Block]*)

**Return type**
bool

angr.analyses.decompiler.utils.**find_block_by_addr**(*graph*, *addr*, *insn_addr=False*)

**Return type**
Block

**Parameters**
**graph** (*DiGraph*)

angr.analyses.decompiler.utils.**find_block_by_addr_and_idx**(*graph*, *addr*, *idx*)

**Return type**
Block

**Parameters**

- **graph** (*DiGraph*)
- **addr** (*int*)
- **idx** (*int | None*)

angr.analyses.decompiler.utils.**sequence_to_blocks**(*seq*)

Converts a sequence node (BaseNode) to a list of ailment blocks contained in it and all its children.

**Return type**
list[Block]

**Parameters**
**seq** (BaseNode)

angr.analyses.decompiler.utils.**sequence_to_statements**(*seq*, *exclude=(<class 'angr.ailment.statement.Jump'>, <class 'angr.ailment.statement.Jump'>)*)

Converts a sequence node (BaseNode) to a list of ailment Statements contained in it and all its children. May exclude certain types of statements.

**Return type**
list[*Statement*]

**Parameters**
**seq** (BaseNode)

angr.analyses.decompiler.utils.**remove_edges_in_ailgraph**(*ail_graph*, *edges_to_remove*)

**Return type**
None

**Parameters**

- **ail_graph** (*DiGraph*)
- **edges_to_remove** (*list[tuple[tuple[int, int | None], tuple[int, int | None]]]*)

### angr.analyses.decompiler.variable_map

angr.analyses.decompiler.variable_map.**variable_map_of**(*manager*)

Return the *VariableMap* attached to an ailment `Manager`, lazily creating and attaching an empty one if the manager does not have a map yet (e.g. Managers constructed outside of Clinic in tests). This keeps consumers that reach the map through `manager.variable_map` from having to special-case `None`.

**Return type**
*VariableMap*

**Parameters**
**manager** (*Manager*)

**class** angr.analyses.decompiler.variable_map.**VariableMap**

Bases: `object`

A side container that maps the `.idx` of AIL `Statement` and `Expression` objects to variable-related information.

The following pieces of information are tracked:

- `variable` (a `SimVariable`) and `variable_offset` (an `int`): the variable that an AIL atom resolves to, and the offset into that variable.
- `custom_string` (a `bool`): whether a `Const` expression refers to a custom string.
- `reference_values` (a `dict` mapping `SimType` to a value): reference values associated with a `Const` expression (e.g., custom strings).
- `reference_variable` (a `SimVariable`) and `reference_variable_offset` (an `int`): the variable that a constant expression references, and the offset into it. These are siblings of `variable` / `variable_offset` that are specifically used for constants that reference global/extern variables.
- `prototype` (a `SimTypeFunction`) and `calling_convention` (a `SimCC`): the call-site prototype and calling convention associated with an AIL `Call` expression. These used to live directly on the `Call` object; they are heavy, non-serializable Python references, so they are tracked here instead.
- `variant` (an `EnumVariant`): the enum variant that a Rust `Let` expression binds.
- `returnty` (a `SimType`): the return type of a Rust `FunctionLikeMacro` call.

Keys are the integer `.idx` values of AIL Statement/Expression objects. Because `Clinic` builds one `ailment.Manager` per invocation, `.idx` values are unique within a single Clinic. So a VariableMap is scoped to one Clinic instance and is stored in the corresponding `DecompilationCache`.

**__init__**()

variable(*obj*)

**Return type**
*SimVariable* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

variable_offset(*obj*)

**Return type**
int

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

custom_string(*obj*)

**Return type**
bool

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

reference_values(*obj*)

**Return type**
dict[*SimType*, Any] | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

reference_variable(*obj*)

**Return type**
*SimVariable* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

reference_variable_offset(*obj*)

**Return type**
int

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

has_variable(*obj*)

**Return type**
bool

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

prototype(*obj*)

**Return type**
*SimTypeFunction* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**calling_convention**(*obj*)

**Return type**
*SimCC* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**variant**(*obj*)

**Return type**
Any

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**returnty**(*obj*)

**Return type**
*SimType* | None

**Parameters**
**obj** (TaggedObject | *Expression* | *Statement* | *int*)

**set_variable**(*obj*, *variable*, *offset=0*)

Set the variable information for an AIL atom. If `variable` is `None`, the variable information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **variable** (SimVariable | *None*)
- **offset** (*int*)

**set_variable_offset**(*obj*, *offset*)

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **offset** (*int*)

**set_custom_string**(*obj*, *value=True*)

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **value** (*bool*)

**set_reference_values**(*obj*, *reference_values*)

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **reference_values** (*dict*[SimType, *Any*])

**set_reference_variable**(*obj*, *variable*, *offset=0*)

Set the reference variable information for an AIL atom. If `variable` is `None`, the reference variable information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **variable** (SimVariable | *None*)
- **offset** (*int*)

**set_prototype**(*obj*, *prototype*)

Set the call-site prototype for an AIL Call. If `prototype` is `None`, the prototype information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **prototype** (SimTypeFunction | *None*)

**set_calling_convention**(*obj*, *cc*)

Set the calling convention for an AIL Call. If `cc` is `None`, the calling-convention information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **cc** (SimCC | *None*)

**set_variant**(*obj*, *variant*)

Set the enum variant for a Rust `Let` expression. If `variant` is `None`, the variant information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **variant** (*Any*)

**set_returnty**(*obj*, *returnty*)

Set the return type for a Rust `FunctionLikeMacro` call. If `returnty` is `None`, the return-type information for this atom is cleared.

**Return type**
None

**Parameters**

- **obj** (TaggedObject | *Expression* | *Statement* | *int*)
- **returnty** (SimType | *None*)

**transfer**(*src*, *dst*)

Copy all variable information associated with `src` to `dst`. Used when an AIL atom is deep-copied to a new `.idx` (e.g. during structuring/duplication) so that the new atom keeps the same variable association.

**Return type**
None

**Parameters**

- **src** (TaggedObject | *int*)
- **dst** (TaggedObject | *int*)

**to_json**()

Serialize this VariableMap to a JSON-compatible object.

Variables are referenced by their `.ident` (reference-by-ident); they must be resolved back to `SimVariable` objects via a resolver in *from_json()*.

**Return type**
dict[str, Any]

**classmethod from_json**(*data*, *resolve_variable*)

Deserialize a VariableMap from a JSON-compatible object produced by *to_json()*.

**Parameters**

- **data** (dict[str, Any]) – The JSON object.
- **resolve_variable** (Callable[[str], *SimVariable* | None]) – A callable that maps a variable ident (`str`) to a `SimVariable` (or `None` if it cannot be resolved).

**Return type**
*VariableMap*

### 10.3.20 angr.analyses.deobfuscator

**class** angr.analyses.deobfuscator.**APIObfType1PeepholeOptimizer**

Bases: *PeepholeOptimizationExprBase*

Integrate type-1 deobfuscated API into decompilation output.

**NAME = 'Simplify Type 1 API obfuscation references'**

**expr_classes = (<class 'angr.ailment.expression.Load'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Load)

class angr.analyses.deobfuscator.**APIObfType3PeepholeOptimizer**

Bases: *PeepholeOptimizationExprBase*

Integrate type-3 deobfuscated APIs (calls with const arguments which return APIs) into the decompilation output.

**NAME = 'Simplify Type 3 API obfuscation references'**

**expr_classes = (<class 'angr.ailment.expression.Call'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Call)

class angr.analyses.deobfuscator.**APIObfuscationFinder**

Bases: *Analysis*

An analysis that automatically finds API "obfuscation" routines.

Currently, we support the following API "obfuscation" styles:

- Type 1: sub_A("dll_name", "api_name") where sub_A ends up calling LoadLibrary.
- Type 2: GetProcAddress(_, "api_name").

**__init__**(*decompiler_options=None*)

**Parameters**
**decompiler_options** (list[tuple[*DecompilationOption* | str, Any]] | None) – Options for the decompilations that the hash-lookup API deobfuscator runs.

**analyze**()

**static is_libname**(*name*)

**Return type**
bool

**Parameters**
**name** (*str*)

**static is_apiname**(*name*)

**Return type**
bool

**Parameters**
**name** (*str*)

class angr.analyses.deobfuscator.**DataTransformationEmbedder**

Bases: *Analysis*

An analysis that finds potentially inlined static data transformation logic and embeds the transformed data in decompilation whenever possible.

Some current limitations: - We assume the data transformation logic is inlined completely within a single function.

**__init__**(*func*, *clinic*, *cfunc*, *outlining_max_args=1*, *preset='malware'*)

**Parameters**

- **func** (Function)

- **clinic** (Clinic)
- **cfunc** (CFunction | *None*)
- **outlining_max_args** (*int*)
- **preset** (*str*)

**class** angr.analyses.deobfuscator.**HashLookupAPIDeobfuscator**

Bases: *Analysis*

An analysis that finds functions accessing loader metadata which take concrete arguments and executes them to see if they resolve symbols.

**__init__**(*lifter*, *func_addrs=None*)

**Parameters**

- **lifter** (*Callable[[Function], Clinic | None]*)
- **func_addrs** (*Sequence[int] | None*)

**class** angr.analyses.deobfuscator.**StringObfType1PeepholeOptimizer**

Bases: *PeepholeOptimizationExprBase*

Integrate type-1 deobfuscated strings into decompilation output.

**NAME = 'Simplify Type 1/2 string deobfuscation references'**

**expr_classes = (<class 'angr.ailment.expression.Call'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**

**expr** (Call)

**class** angr.analyses.deobfuscator.**StringObfType3Rewriter**

Bases: *OptimizationPass*

Type-3 optimization pass replaces deobfuscate_string calls with the deobfuscated strings, and then removes arguments on the stack.

**ARCHES = ['X86', 'AMD64']**

**PLATFORMS = ['windows']**

**STAGE: *OptimizationPassStage* = 5**

**NAME = 'Simplify Type 3 string deobfuscation calls'**

**DESCRIPTION = 'Simplify Type 3 string deobfuscation calls'**

**stmt_classes = ()**

**__init__**(**args*, ***kwargs*)

**static is_call_or_call_assignment**(*stmt*)

**Return type**

bool

class angr.analyses.deobfuscator.**StringObfuscationFinder**

Bases: *Analysis*

An analysis that automatically finds string obfuscation routines.

**__init__**(*functions=None*)

**Parameters**
**functions** (*list[Function] | None*)

**analyze**()

### Submodules


*api_obf_finder*
*api_obf_peephole_optimizer*
*api_obf_type2_finder*
*data_transformation_embedder*
*hash_lookup_api_deobfuscator*
*irsb_reg_collector*
*scope_ops_analyzer*
*string_obf_finder*
*string_obf_opt_passes*
*string_obf_peephole_optimizer*


### angr.analyses.deobfuscator.api_obf_finder

class angr.analyses.deobfuscator.api_obf_finder.**APIObfuscationType**

Bases: IntEnum

**TYPE_1 = 0**

**__new__**(*value*)

class angr.analyses.deobfuscator.api_obf_finder.**APIDeobFuncDescriptor**

Bases: object

**__init__**(*type_*, ***, *func_addr*, *libname_argidx*, *funcname_argidx*)

**Parameters**

- **type_** (APIObfuscationType)
- **func_addr** (*int*)
- **libname_argidx** (*int*)
- **funcname_argidx** (*int*)

class angr.analyses.deobfuscator.api_obf_finder.**Type1AssignmentFinder**

Bases: *CStructuredCodeWalker*

**__init__**(*func_addr*, *desc*)

**Parameters**

- **func_addr** (*int*)
- **desc** (APIDeobFuncDescriptor)

**assignments**: dict[int, tuple[str, str]]

**handle_CAssignment**(*obj*)

**Parameters**
**obj** (CAssignment)

class angr.analyses.deobfuscator.api_obf_finder.**APIObfuscationFinder**

Bases: *Analysis*

An analysis that automatically finds API "obfuscation" routines.

Currently, we support the following API "obfuscation" styles:

- Type 1: sub_A("dll_name", "api_name") where sub_A ends up calling LoadLibrary.
- Type 2: GetProcAddress(_, "api_name").

**__init__**(*decompiler_options=None*)

**Parameters**
**decompiler_options** (list[tuple[*DecompilationOption* | str, Any]] | None) – Options for the decompilations that the hash-lookup API deobfuscator runs.

**analyze**()

**static is_libname**(*name*)

**Return type**
bool

**Parameters**
**name** (*str*)

**static is_apiname**(*name*)

**Return type**
bool

**Parameters**
**name** (*str*)

### angr.analyses.deobfuscator.api_obf_peephole_optimizer

class angr.analyses.deobfuscator.api_obf_peephole_optimizer.**APIObfType1PeepholeOptimizer**

Bases: *PeepholeOptimizationExprBase*

Integrate type-1 deobfuscated API into decompilation output.

**NAME = 'Simplify Type 1 API obfuscation references'**

**expr_classes = (<class 'angr.ailment.expression.Load'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Load)

class angr.analyses.deobfuscator.api_obf_peephole_optimizer.**APIObfType3PeepholeOptimizer**

Bases: *PeepholeOptimizationExprBase*

Integrate type-3 deobfuscated APIs (calls with const arguments which return APIs) into the decompilation output.

**NAME = 'Simplify Type 3 API obfuscation references'**

**expr_classes = (<class 'angr.ailment.expression.Call'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Call)

### angr.analyses.deobfuscator.api_obf_type2_finder

**class** angr.analyses.deobfuscator.api_obf_type2_finder.**APIObfuscationType2**

Bases: object

API Obfuscation Type 2 result.

**resolved_func_name: str**

**resolved_func_ptr: *MemoryLocation***

**resolved_in: *Function***

**resolved_by: *Function***

**__init__**(*resolved_func_name*, *resolved_func_ptr*, *resolved_in*, *resolved_by*)

**Parameters**

- **resolved_func_name** (*str*)
- **resolved_func_ptr** (MemoryLocation)
- **resolved_in** (Function)
- **resolved_by** (Function)

**Return type**
None

**class** angr.analyses.deobfuscator.api_obf_type2_finder.**APIObfuscationType2Finder**

Bases: object

Finds global function pointers initialized by calls to dlsym/GetProcAddress and names them accordingly.

**__init__**(*project*, *kb=None*)

**Parameters**

- **project** (Project)
- **kb** (KnowledgeBase | *None*)

**results: list[*APIObfuscationType2*]**

**analyze**()

**Return type**
list[*APIObfuscationType2*]

### angr.analyses.deobfuscator.data_transformation_embedder

**class** angr.analyses.deobfuscator.data_transformation_embedder.**DataTransformationEmbedder**

Bases: *Analysis*

An analysis that finds potentially inlined static data transformation logic and embeds the transformed data in decompilation whenever possible.

Some current limitations: - We assume the data transformation logic is inlined completely within a single function.

**__init__**(*func*, *clinic*, *cfunc*, *outlining_max_args=1*, *preset='malware'*)

**Parameters**

- **func** (Function)
- **clinic** (Clinic)
- **cfunc** (*CFunction | None*)
- **outlining_max_args** (*int*)
- **preset** (*str*)

### angr.analyses.deobfuscator.hash_lookup_api_deobfuscator

**class** angr.analyses.deobfuscator.hash_lookup_api_deobfuscator.**HashLookupAPIDeobfuscator**

Bases: *Analysis*

An analysis that finds functions accessing loader metadata which take concrete arguments and executes them to see if they resolve symbols.

**__init__**(*lifter*, *func_addrs=None*)

**Parameters**

- **lifter** (*Callable[[Function], Clinic | None]*)
- **func_addrs** (*Sequence[int] | None*)

**class** angr.analyses.deobfuscator.hash_lookup_api_deobfuscator.**FindCallsTo**

Bases: *AILBlockViewer*

Walker which stores calls with a given target.

**__init__**(**args*, *target*, *variable_map=None*, ***kwargs*)

**Parameters**
**target** (*str | int*)

### angr.analyses.deobfuscator.irsb_reg_collector

**class** angr.analyses.deobfuscator.irsb_reg_collector.**IRSBRegisterCollector**

Bases: *SimEngineNostmtVEX*[None, None, None]

Scan the VEX IRSB to collect all registers that are read.

**__init__**(**args*, ***kwargs*)

### angr.analyses.deobfuscator.scope_ops_analyzer

**class** angr.analyses.deobfuscator.scope_ops_analyzer.**ScopeOpsWalker**

Bases: *CStructuredCodeWalker*

ScopeOpsWalker walks a C construct (typically a C function) and extracts counts of all operations in each scope. Its intended use is to reason about the intent of different parts of functions (e.g., crypto implementations).

**__init__**()

**handle**(*obj*)

**handle_CBinaryOp**(*obj*)

**handle_CUnaryOp**(*obj*)

**class** angr.analyses.deobfuscator.scope_ops_analyzer.**ScopeOpsAnalyzer**

Bases: *Analysis*

An analysis that extracts and analyzes operations used by different scopes of a function.

**__init__**(*cfunc*)

**Parameters**
**cfunc** (CFunction)

**filter_scopes**(*wanted_ops*, *min_count*)

**Parameters**
**wanted_ops** (set)

**crypto_scopes**(*min_count=20*)

### angr.analyses.deobfuscator.string_obf_finder

**class** angr.analyses.deobfuscator.string_obf_finder.**StringDeobFuncDescriptor**

Bases: object

Describes a string deobfuscation function.

**string_input_arg_idx:** int

**string_output_arg_idx:** int

**__init__**()

**string_length_arg_idx:** int | None

**string_null_terminating:** bool | None

**class** angr.analyses.deobfuscator.string_obf_finder.**StringObfuscationFinder**

Bases: *Analysis*

An analysis that automatically finds string obfuscation routines.

**__init__**(*functions=None*)

**Parameters**
**functions** (*list[Function] | None*)

**analyze**()

#### angr.analyses.deobfuscator.string_obf_opt_passes

**class** angr.analyses.deobfuscator.string_obf_opt_passes.**StringObfType3Rewriter**

Bases: *OptimizationPass*

Type-3 optimization pass replaces deobfuscate_string calls with the deobfuscated strings, and then removes arguments on the stack.

**ARCHES = ['X86', 'AMD64']**

**PLATFORMS = ['windows']**

**STAGE: *OptimizationPassStage* = 5**

**NAME = 'Simplify Type 3 string deobfuscation calls'**

**DESCRIPTION = 'Simplify Type 3 string deobfuscation calls'**

**stmt_classes = ()**

**__init__**(**args*, ***kwargs*)

**static is_call_or_call_assignment**(*stmt*)

**Return type**
bool

#### angr.analyses.deobfuscator.string_obf_peephole_optimizer

**class**
angr.analyses.deobfuscator.string_obf_peephole_optimizer.**StringObfType1PeepholeOptimizer**

Bases: *PeepholeOptimizationExprBase*

Integrate type-1 deobfuscated strings into decompilation output.

**NAME = 'Simplify Type 1/2 string deobfuscation references'**

**expr_classes = (<class 'angr.ailment.expression.Call'>,)**

**optimize**(*expr*, ***kwargs*)

**Parameters**
**expr** (Call)

### 10.3.21 angr.analyses.disassembly

**class** angr.analyses.disassembly.**DisassemblyPiece**

Bases: object

**addr = None**

**ident = nan**

**render**(*formatting=None*)

**Return type**
list[str]

**getpiece**(*formatting*, *column*)

**width**(*formatting*)

**height**(*formatting*)

**static color**(*string*, *coloring*, *formatting*)

**highlight**(*string*, *formatting=None*)

**class** angr.analyses.disassembly.**FunctionStart**

Bases: *DisassemblyPiece*

**__init__**(*func*)

Constructor.

**Parameters**

**func** (*angr.knowledge.Function*) – The function instance.

**height**(*formatting*)

**class** angr.analyses.disassembly.**Label**

Bases: *DisassemblyPiece*

**__init__**(*addr*, *name*)

**class** angr.analyses.disassembly.**IROp**

Bases: *DisassemblyPiece*

**__init__**(*addr*, *seq*, *obj*, *irsb*)

**Parameters**

- **addr** (*int*)
- **seq** (*int*)
- **obj** (*IRStmt | PcodeOp*)
- **irsb** (*IRSB | IRSB*)

**addr:** int

**seq:** int

**obj:** IRStmt | PcodeOp

**irsb:** IRSB | *IRSB*

**class** angr.analyses.disassembly.**BlockStart**

Bases: *DisassemblyPiece*

**__init__**(*block*, *parentfunc*, *project*)

**class** angr.analyses.disassembly.**Hook**

Bases: *DisassemblyPiece*

**__init__**(*block*)

**class** angr.analyses.disassembly.**Instruction**

Bases: *DisassemblyPiece*

**__init__**(*insn*, *parentblock*, *project=None*)

opcode: *Opcode*

property mnemonic

reload_format()

dissect_instruction()

dissect_instruction_for_arm()

static split_arm_op_string(*op_str*)

**Parameters**
**op_str** (*str*)

dissect_instruction_by_default()

static split_op_string(*insn_str*)

class angr.analyses.disassembly.**SootExpression**

Bases: *DisassemblyPiece*

__init__(*expr*)

class angr.analyses.disassembly.**SootExpressionTarget**

Bases: *SootExpression*

__init__(*target_stmt_idx*)

class angr.analyses.disassembly.**SootExpressionStaticFieldRef**

Bases: *SootExpression*

__init__(*field*)

class angr.analyses.disassembly.**SootExpressionInvoke**

Bases: *SootExpression*

Virtual = 'virtual'

Static = 'static'

Special = 'special'

__init__(*invoke_type*, *expr*)

class angr.analyses.disassembly.**SootStatement**

Bases: *DisassemblyPiece*

__init__(*block_addr*, *raw_stmt*)

property stmt_idx

class angr.analyses.disassembly.**Opcode**

Bases: *DisassemblyPiece*

__init__(*parentinsn*)

class angr.analyses.disassembly.**Operand**

Bases: *DisassemblyPiece*

__init__(*op_num*, *children*, *parentinsn*)

children: list

property cs_operand

static build(*operand_type*, *op_num*, *children*, *parentinsn*)

class angr.analyses.disassembly.ConstantOperand

Bases: *Operand*

class angr.analyses.disassembly.RegisterOperand

Bases: *Operand*

property register

class angr.analyses.disassembly.MemoryOperand

Bases: *Operand*

__init__(*op_num*, *children*, *parentinsn*)

values: list[str | *DisassemblyPiece*]

class angr.analyses.disassembly.OperandPiece

Bases: *DisassemblyPiece*

addr = None

parentop = None

ident = None

class angr.analyses.disassembly.Register

Bases: *OperandPiece*

__init__(*reg*, *prefix=''*)

class angr.analyses.disassembly.Value

Bases: *OperandPiece*

__init__(*val*, *render_with_sign*, *render_with_pound_sign=False*)

**Parameters**

**render_with_pound_sign** (*bool*)

property project

class angr.analyses.disassembly.Comment

Bases: *DisassemblyPiece*

__init__(*addr*, *text*)

height(*formatting*)

class angr.analyses.disassembly.FuncComment

Bases: *DisassemblyPiece*

__init__(*func*)

**class** angr.analyses.disassembly.**Disassembly**

Bases: *Analysis*

Produce formatted machine code disassembly.

**__init__**(*function=None*, *ranges=None*, *thumb=False*, *include_ir=False*, *block_bytes=None*)

**Parameters**

- **function** (*Function | None*)
- **ranges** (*Sequence[tuple[int, int]] | None*)
- **thumb** (*bool*)
- **include_ir** (*bool*)
- **block_bytes** (*bytes | None*)

**func_lookup**(*block*)

**parse_block**(*block*)

Parse instructions for a given block node

**Return type**
None

**Parameters**
**block** (BlockNode)

**render**(*formatting=None*, *show_edges=True*, *show_addresses=True*, *show_bytes=False*, *ascii_only=None*, *color=True*, *min_edge_depth=0*)

Render the disassembly to a string, with optional edges and addresses.

Color will be added by default, if enabled. To disable color pass an empty formatting dict.

**Return type**
str

**Parameters**

- **show_edges** (*bool*)
- **show_addresses** (*bool*)
- **show_bytes** (*bool*)
- **ascii_only** (*bool | None*)
- **color** (*bool*)
- **min_edge_depth** (*int*)

### 10.3.22 angr.analyses.disassembly_utils

angr.analyses.disassembly_utils.**decode_instruction**(*arch*, *instr*)

### 10.3.23 angr.analyses.dominance_frontier

**class** angr.analyses.dominance_frontier.**DominanceFrontier**

Bases: Generic

Computes the dominance frontier of all nodes in a function graph, and provides an easy-to-use interface for querying the frontier information.

__init__(*func*, *func_graph=None*, *entry=None*, *exception_edges=False*)

**Overloads**

- **self**, **func** (Function), **func_graph** (networkx.DiGraph[T_co]), **entry** (T_co), **exception_edges** (bool)
- **self** (DominanceFrontier[CodeNode]), **func** (Function), **func_graph** (networkx.DiGraph[CodeNode] | None), **entry** (CodeNode | None), **exception_edges** (bool)

angr.analyses.dominance_frontier.**calculate_iterated_dominace_frontier_set**(*frontiers*, *blocks*)

**Return type**
set

**Parameters**

- **frontiers** (*dict*)
- **blocks** (*set*)

### 10.3.24 angr.analyses.fcp

**class** angr.analyses.fcp.**FastConstantPropagation**

Bases: *Analysis*

An extremely fast constant propagation analysis that finds function-wide constant values with potentially high false negative rates.

__init__(*func*, *blocks=None*, *vex_cross_insn_opt=False*, *load_callback=None*)

**Parameters**

- **func** (Function)
- **blocks** (*set[Block] | None*)
- **vex_cross_insn_opt** (*bool*)
- **load_callback** (*Callable | None*)

#### Submodules

*fcp*

#### angr.analyses.fcp.fcp

**class** angr.analyses.fcp.fcp.**SV**

Bases: object

SizedValue: A faster implementation of claripy.ast.BV.

__init__(*value*, *bits*)

**value**

**bits**

class angr.analyses.fcp.fcp.**FCPState**

Bases: object

The abstract state for FastConstantPropagation.

**__init__**()

**tmps**

**simple_stack**

**regs**: dict[int, *SV*]

**stack**: dict[int, *SV*]

**sp_value**

**bp_value**

**register_read**(*offset*, *size_in_bytes*)

**Return type**
int | None

**Parameters**
**size_in_bytes** (*int*)

**register_written**(*offset*, *size_in_bytes*, *value*)

**Parameters**

- **offset** (*int*)
- **size_in_bytes** (*int*)
- **value** (*int | None*)

**stack_read**(*offset*, *size_int_bytes*)

**Return type**
int | None

**Parameters**

- **offset** (*int*)
- **size_int_bytes** (*int*)

**stack_written**(*offset*, *size_int_bytes*, *value*)

**Parameters**

- **offset** (*int*)
- **size_int_bytes** (*int*)
- **value** (*int | None*)

**copy**(*with_tmps=False*)

**Return type**
*FCPState*

**Parameters**
**with_tmps** (*bool*)

**callee_stored_regs**

**class** angr.analyses.fcp.fcp.**SimEngineFCPVEX**

Bases: *SimEngineNostmtVEX*[*FCPState*, *SpOffset* | *RegisterOffset* | int, None], *SimEngineLight*[type[*FCPState*], *SpOffset* | *RegisterOffset* | int, *Block*, None]

THe engine for FastConstantPropagation.

**__init__**(*project*, *bp_as_gpr*, *replacements*, *load_callback=None*)

**Parameters**

- **bp_as_gpr** (*bool*)
- **replacements** (*dict[int, dict]*)
- **load_callback** (*Callable | None*)

**class** angr.analyses.fcp.fcp.**FastConstantPropagation**

Bases: *Analysis*

An extremely fast constant propagation analysis that finds function-wide constant values with potentially high false negative rates.

**__init__**(*func*, *blocks=None*, *vex_cross_insn_opt=False*, *load_callback=None*)

**Parameters**

- **func** (Function)
- **blocks** (*set[Block] | None*)
- **vex_cross_insn_opt** (*bool*)
- **load_callback** (*Callable | None*)

### 10.3.25 angr.analyses.find_objects_static

**class** angr.analyses.find_objects_static.**PossibleObject**

Bases: object

This holds the address and class name of possible class instances. The address that it holds in mapped outside the binary so it is only valid in this analysis. TO DO: map the address to its uses in the registers/memory locations in the instructions

**__init__**(*size*, *addr*, *class_name=None*)

**class** angr.analyses.find_objects_static.**NewFunctionHandler**

Bases: *FunctionHandler*

**This handles calls to the function new(), by recording the size parameter passed to it and also assigns a new**
address outside the mapped binary to the newly created space(possible object).

It also tracks if the function called right after new() is passed the same 'this' pointer and is a constructor, if so we mark it as an instance of the class the constructor belongs to.(only for non stripped binaries)

**handle_local_function**(*state*, *data*)

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

class angr.analyses.find_objects_static.**StaticObjectFinder**

Bases: *Analysis*

**This analysis tries to find objects on the heap based on calls to new(), and subsequent calls to constructors with**
the 'this' pointer

**__init__**()

### 10.3.26 angr.analyses.flirt

class angr.analyses.flirt.**FlirtAnalysis**

Bases: *Analysis*

FlirtAnalysis accomplishes two purposes:

- If a FLIRT signature file is specified, it will match the given signature file against the current binary and rename recognized functions accordingly.
- If no FLIRT signature file is specified, it will use strings to determine possible libraries embedded in the current binary, and then match all possible signatures for the architecture.

**__init__**(*sig=None*, *max_mismatched_bytes=0*, *dry_run=False*, *match_named_functions=False*)

**Parameters**

- **sig** (*FlirtSignature | str | None*)
- **max_mismatched_bytes** (*int*)
- **dry_run** (*bool*)
- **match_named_functions** (*bool*)

class angr.analyses.flirt.**FlirtSignature**

Bases: object

This class describes a FLIRT signature without any internal data that is only available after parsing.

**__init__**(*arch*, *platform*, *sig_name*, *sig_path*, *unique_strings=None*, *compiler=None*, *compiler_version=None*, *os_name=None*, *os_version=None*)

**Parameters**

- **arch** (*str*)
- **platform** (*str*)
- **sig_name** (*str*)
- **sig_path** (*str*)
- **unique_strings** (*set[str] | None*)
- **compiler** (*str | None*)
- **compiler_version** (*str | None*)
- **os_name** (*str | None*)
- **os_version** (*str | None*)

**exception** angr.analyses.flirt.**FlirtSignatureError**

Bases: *AngrError*

Describes errors related to FLIRT signatures, especially parsing.

**class** angr.analyses.flirt.**FlirtSignatureParsed**

Bases: object

Describes a FLIRT signature file after parsing.

**__init__**(*version*, *arch*, *file_types*, *os_types*, *app_types*, *features*, *crc*, *ctype*, *ctypes_crc*, *nfuncs*, *pattern_size*, *libname*, *root*)

**Parameters**

- **version** (*int*)
- **arch** (*int*)
- **file_types** (*int*)
- **os_types** (*int*)
- **app_types** (*int*)
- **features** (*int*)
- **crc** (*int*)
- **ctype** (*int*)
- **ctypes_crc** (*int*)
- **nfuncs** (*int | None*)
- **pattern_size** (*int | None*)
- **libname** (*str*)
- **root** (FlirtNode | *None*)

**version**

**arch**

**file_types**

**os_types**

**app_types**

**features**

**crc**

**ctype**

**ctypes_crc**

**nfuncs**

**pattern_size**

**libname**

**root**

**parse_tree**(*file_obj*, *root=False*)

Parse a FLIRT function tree.

**Return type**
*FlirtNode*

**Parameters**
**root** (*bool*)

**parse_public_function**(*file_obj*, *offset*)

**Return type**
tuple[*FlirtFunction*, int, int]

**Parameters**
**offset** (*int*)

**parse_referenced_functions**(*file_obj*)

**Return type**
list[*FlirtFunction*]

**parse_tail_bytes**(*file_obj*)

**Return type**
list[tuple[int, int]]

**parse_modules**(*file_obj*)

**Return type**
list[*FlirtModule*]

**parse_module**(*file_obj*)

**Return type**
tuple[*FlirtModule*, int]

**static parse_variant_mask**(*file_obj*, *length*)

**Return type**
int

**Parameters**
**length** (*int*)

**static is_bit_set_be**(*mask*, *mask_len*, *bit_offset*)

**Return type**
bool

**Parameters**

- **mask** (*int*)
- **mask_len** (*int*)
- **bit_offset** (*int*)

**static parse_node**(*file_obj*, *length*, *variant_mask*)

**Return type**
list[int]

**Parameters**

- **length** (*int*)
- **variant_mask** (*int*)

**classmethod parse_header**(*file_obj*)

Parse only the FLIRT signature header (no function tree).

The returned object has `root` set to None. Use `parse()` if you also need the function tree, or call `parse_tree_from()` later on a separately opened file to populate `root` on demand.

The following struct definitions come from radare2

// FLIRT v5+ ut8 magic[6]; ut8 version; ut8 arch; ut32 file_types; ut16 os_types; ut16 app_types; ut16 features; ut16 old_n_functions; ut16 crc16; ut8 ctype[12]; ut8 library_name_len; ut16 ctypes_crc16;

// FLIRT v6+ ut32 nfuncs;

// FLIRT v8+ ut16 pattern_size;

// FLIRT v10 ut16 unknown;

**Return type**
*FlirtSignatureParsed*

**classmethod parse**(*file_obj*)

Parse a FLIRT signature file, including the function tree.

**Return type**
*FlirtSignatureParsed*

angr.analyses.flirt.**flirt_arch_to_arch_name**(*flirt_arch*, *app_types*)

Convert FLIRT architecture ID to architecture name.

**Parameters**

- **flirt_arch** (int) – FLIRT architecture ID.
- **app_types** (int) – FLIRT application types.

**Return type**
str

**Returns**
Architecture name.

angr.analyses.flirt.**flirt_os_type_to_os_name**(*os_type*)

Convert FLIRT OS type to OS name.

**Parameters**
**os_type** (int) – FLIRT OS type.

**Return type**
str

**Returns**
OS name.

## Submodules

*consts*

continues on next page

Table 28 – continued from previous page

| |
|---|
| *flirt* |
| *flirt_function* |
| *flirt_matcher* |
| *flirt_module* |
| *flirt_node* |
| *flirt_sig* |
| *flirt_utils* |

### angr.analyses.flirt.consts

**class** angr.analyses.flirt.consts.**FlirtArch**

Bases: int, Enum

**ARCH_386 = 0**

**ARCH_Z80 = 1**

**ARCH_I860 = 2**

**ARCH_8051 = 3**

**ARCH_TMS = 4**

**ARCH_6502 = 5**

**ARCH_PDP = 6**

**ARCH_68K = 7**

**ARCH_JAVA = 8**

**ARCH_6800 = 9**

**ARCH_ST7 = 10**

**ARCH_MC6812 = 11**

**ARCH_MIPS = 12**

**ARCH_ARM = 13**

**ARCH_TMSC6 = 14**

**ARCH_PPC = 15**

**ARCH_80196 = 16**

**ARCH_Z8 = 17**

**ARCH_SH = 18**

**ARCH_NET = 19**

**ARCH_AVR = 20**

**ARCH_H8 = 21**

**ARCH_PIC = 22**

```
ARCH_SPARC = 23

ARCH_ALPHA = 24

ARCH_HPPA = 25

ARCH_H8500 = 26

ARCH_TRICORE = 27

ARCH_DSP56K = 28

ARCH_C166 = 29

ARCH_ST20 = 30

ARCH_IA64 = 31

ARCH_I960 = 32

ARCH_F2MC = 33

ARCH_TMS320C54 = 34

ARCH_TMS320C55 = 35

ARCH_TRIMEDIA = 36

ARCH_M32R = 37

ARCH_NEC_78K0 = 38

ARCH_NEC_78K0S = 39

ARCH_M740 = 40

ARCH_M7700 = 41

ARCH_ST9 = 42

ARCH_FR = 43

ARCH_MC6816 = 44

ARCH_M7900 = 45

ARCH_TMS320C3 = 46

ARCH_KR1878 = 47

ARCH_AD218X = 48

ARCH_OAKDSP = 49

ARCH_TLCS900 = 50

ARCH_C39 = 51

ARCH_CR16 = 52

ARCH_MN102L00 = 53
```

ARCH_TMS320C1X = 54

ARCH_NEC_V850X = 55

ARCH_SCR_ADPT = 56

ARCH_EBC = 57

ARCH_MSP430 = 58

ARCH_SPU = 59

ARCH_DALVIK = 60

__new__(*value*)

class angr.analyses.flirt.consts.**FlirtFileType**

Bases: int, Enum

FILE_DOS_EXE_OLD = 1

FILE_DOS_COM_OLD = 2

FILE_BIN = 4

FILE_DOSDRV = 8

FILE_NE = 16

FILE_INTELHEX = 32

FILE_MOSHEX = 64

FILE_LX = 128

FILE_LE = 256

FILE_NLM = 512

FILE_COFF = 1024

FILE_PE = 2048

FILE_OMF = 4096

FILE_SREC = 8192

FILE_ZIP = 16384

FILE_OMFLIB = 32768

FILE_AR = 65536

FILE_LOADER = 131072

FILE_ELF = 262144

FILE_W32RUN = 524288

FILE_AOUT = 1048576

FILE_PILOT = 2097152

FILE_DOS_EXE = 4194304

FILE_DOS_COM = 8388608

FILE_AIXAR = 16777216

__new__(*value*)

class angr.analyses.flirt.consts.**FlirtOSType**

Bases: int, Enum

Actually no longer used in IDA.

OS_MSDOS = 1

OS_WIN = 2

OS_OS2 = 4

OS_NETWARE = 8

OS_UNIX = 16

OS_OTHER = 32

__new__(*value*)

class angr.analyses.flirt.consts.**FlirtAppType**

Bases: int, Enum

APP_CONSOLE = 1

APP_GRAPHICS = 2

APP_EXE = 4

APP_DLL = 8

APP_DRV = 16

APP_SINGLE_THREADED = 32

APP_MULTI_THREADED = 64

APP_16_BIT = 128

APP_32_BIT = 256

APP_64_BIT = 512

__new__(*value*)

class angr.analyses.flirt.consts.**FlirtFeatureFlag**

Bases: int, Enum

FEATURE_STARTUP = 1

FEATURE_CTYPE_CRC = 2

**FEATURE_2BYTE_CTYPE = 4**

**FEATURE_ALT_CTYPE_CRC = 8**

**FEATURE_COMPRESSED = 16**

**__new__**(*value*)

**class** angr.analyses.flirt.consts.**FlirtParseFlag**

Bases: int, Enum

**PARSE_MORE_PUBLIC_NAMES = 1**

**PARSE_READ_TAIL_BYTES = 2**

**PARSE_READ_REFERENCED_FUNCTIONS = 4**

**PARSE_MORE_MODULES_WITH_SAME_CRC = 8**

**PARSE_MORE_MODULES = 16**

**__new__**(*value*)

**class** angr.analyses.flirt.consts.**FlirtFunctionFlag**

Bases: int, Enum

**FUNCTION_LOCAL = 2**

**FUNCTION_UNRESOLVED_COLLISION = 8**

**__new__**(*value*)

### angr.analyses.flirt.flirt

**class** angr.analyses.flirt.flirt.**FlirtAnalysis**

Bases: *Analysis*

FlirtAnalysis accomplishes two purposes:

- If a FLIRT signature file is specified, it will match the given signature file against the current binary and rename recognized functions accordingly.
- If no FLIRT signature file is specified, it will use strings to determine possible libraries embedded in the current binary, and then match all possible signatures for the architecture.

**__init__**(*sig=None*, *max_mismatched_bytes=0*, *dry_run=False*, *match_named_functions=False*)

**Parameters**

- **sig** (FlirtSignature | *str* | *None*)
- **max_mismatched_bytes** (*int*)
- **dry_run** (*bool*)
- **match_named_functions** (*bool*)

### angr.analyses.flirt.flirt_function

class angr.analyses.flirt.flirt_function.**FlirtFunction**

Bases: object

Describes a function object in a FLIRT signature.

**__init__**(*name*, *offset*, *local*, *collision*)

**Parameters**

- **name** (*str*)
- **offset** (*int*)
- **local** (*bool*)
- **collision** (*bool*)

**name**

**offset**

**local**

**collision**

### angr.analyses.flirt.flirt_matcher

angr.analyses.flirt.flirt_matcher.**crc16**(*data*)

**Return type**
int

**Parameters**
**data** (bytes)

class angr.analyses.flirt.flirt_matcher.**FlirtMatcher**

Bases: object

A class that matches functions in a binary using FLIRT signatures.

**__init__**(*sig*, *func*, *get_callee_name*, *func_matched*, *mismatch_bytes_tolerance=0*)

**Parameters**

- **sig** (FlirtSignatureParsed)
- **func** (Function)
- **get_callee_name** (*Callable[[Function, int, int, str], str | None]*)
- **func_matched** (*Callable[[Function, int, FlirtFunction], None]*)
- **mismatch_bytes_tolerance** (*int*)

**match_function**(*buff*, *addr*)

**Return type**
bool

**Parameters**

- **buff** (bytes)
- **addr** (*int*)

### angr.analyses.flirt.flirt_module

**class** angr.analyses.flirt.flirt_module.**FlirtModule**

Bases: object

Describes a module in a FLIRT signature.

**__init__**(*length*, *crc_len*, *crc*, *pub_funcs*, *ref_funcs*, *tail_bytes*)

**Parameters**

- **length** (*int*)
- **crc_len** (*int*)
- **crc** (*int*)
- **pub_funcs** (*list[FlirtFunction]*)
- **ref_funcs** (*list[FlirtFunction]*)
- **tail_bytes** (*list[tuple[int, int]]*)

**length**

**crc_len**

**crc**

**pub_funcs**

**ref_funcs**

**tail_bytes**

### angr.analyses.flirt.flirt_node

**class** angr.analyses.flirt.flirt_node.**FlirtNode**

Bases: object

Describes a tree node in the FLIRT signature tree.

**__init__**(*children*, *modules*, *length*, *pattern*)

**Parameters**

- **children** (*list[FlirtNode]*)
- **modules** (*list[FlirtModule]*)
- **length** (*int*)
- **pattern** (*list[int]*)

**children**

**modules**

**length**

**pattern**

**property leaf:** bool

### angr.analyses.flirt.flirt_sig

**exception** angr.analyses.flirt.flirt_sig.**FlirtSignatureError**

Bases: *AngrError*

Describes errors related to FLIRT signatures, especially parsing.

**class** angr.analyses.flirt.flirt_sig.**FlirtSignature**

Bases: object

This class describes a FLIRT signature without any internal data that is only available after parsing.

**__init__**(*arch*, *platform*, *sig_name*, *sig_path*, *unique_strings=None*, *compiler=None*, *compiler_version=None*, *os_name=None*, *os_version=None*)

**Parameters**

- **arch** (*str*)
- **platform** (*str*)
- **sig_name** (*str*)
- **sig_path** (*str*)
- **unique_strings** (*set[str] | None*)
- **compiler** (*str | None*)
- **compiler_version** (*str | None*)
- **os_name** (*str | None*)
- **os_version** (*str | None*)

**class** angr.analyses.flirt.flirt_sig.**FlirtSignatureParsed**

Bases: object

Describes a FLIRT signature file after parsing.

**__init__**(*version*, *arch*, *file_types*, *os_types*, *app_types*, *features*, *crc*, *ctype*, *ctypes_crc*, *nfuncs*, *pattern_size*, *libname*, *root*)

**Parameters**

- **version** (*int*)
- **arch** (*int*)
- **file_types** (*int*)
- **os_types** (*int*)
- **app_types** (*int*)
- **features** (*int*)
- **crc** (*int*)
- **ctype** (*int*)
- **ctypes_crc** (*int*)
- **nfuncs** (*int | None*)
- **pattern_size** (*int | None*)
- **libname** (*str*)

- **root** (FlirtNode | *None*)

**version**

**arch**

**file_types**

**os_types**

**app_types**

**features**

**crc**

**ctype**

**ctypes_crc**

**nfuncs**

**pattern_size**

**libname**

**root**

**parse_tree**(*file_obj*, *root=False*)

Parse a FLIRT function tree.

**Return type**
*FlirtNode*

**Parameters**
**root** (*bool*)

**parse_public_function**(*file_obj*, *offset*)

**Return type**
tuple[*FlirtFunction*, int, int]

**Parameters**
**offset** (*int*)

**parse_referenced_functions**(*file_obj*)

**Return type**
list[*FlirtFunction*]

**parse_tail_bytes**(*file_obj*)

**Return type**
list[tuple[int, int]]

**parse_modules**(*file_obj*)

**Return type**
list[*FlirtModule*]

**parse_module**(*file_obj*)

**Return type**
tuple[*FlirtModule*, int]

**static parse_variant_mask**(*file_obj*, *length*)

**Return type**
int

**Parameters**
**length** (*int*)

**static is_bit_set_be**(*mask*, *mask_len*, *bit_offset*)

**Return type**
bool

**Parameters**

- **mask** (*int*)
- **mask_len** (*int*)
- **bit_offset** (*int*)

**static parse_node**(*file_obj*, *length*, *variant_mask*)

**Return type**
list[int]

**Parameters**

- **length** (*int*)
- **variant_mask** (*int*)

**classmethod parse_header**(*file_obj*)

Parse only the FLIRT signature header (no function tree).

The returned object has `root` set to None. Use *parse()* if you also need the function tree, or call `parse_tree_from()` later on a separately opened file to populate `root` on demand.

The following struct definitions come from radare2

// FLIRT v5+ ut8 magic[6]; ut8 version; ut8 arch; ut32 file_types; ut16 os_types; ut16 app_types; ut16 features; ut16 old_n_functions; ut16 crc16; ut8 ctype[12]; ut8 library_name_len; ut16 ctypes_crc16;

// FLIRT v6+ ut32 nfuncs;

// FLIRT v8+ ut16 pattern_size;

// FLIRT v10 ut16 unknown;

**Return type**
*FlirtSignatureParsed*

**classmethod parse**(*file_obj*)

Parse a FLIRT signature file, including the function tree.

**Return type**
*FlirtSignatureParsed*

#### angr.analyses.flirt.flirt_utils

angr.analyses.flirt.flirt_utils.**read_short**(*file_obj*)

**Return type**
int

angr.analyses.flirt.flirt_utils.**read_word**(*file_obj*)

**Return type**
int

angr.analyses.flirt.flirt_utils.**read_max_2_bytes**(*file_obj*)

**Return type**
int

angr.analyses.flirt.flirt_utils.**read_multiple_bytes**(*file_obj*)

**Return type**
int

### 10.3.27 angr.analyses.forward_analysis

**class** angr.analyses.forward_analysis.**CallGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**callgraph** (*networkx.DiGraph*)

**__init__**(*callgraph*)

**class** angr.analyses.forward_analysis.**ForwardAnalysis**

Bases: Generic

This is my very first attempt to build a static forward analysis framework that can serve as the base of multiple static analyses in angr, including CFG analysis, VFG analysis, DDG, etc.

In short, ForwardAnalysis performs a forward data-flow analysis by traversing a graph, compute on abstract values, and store results in abstract states. The user can specify what graph to traverse, how a graph should be traversed, how abstract values and abstract states are defined, etc.

ForwardAnalysis has a few options to toggle, making it suitable to be the base class of several different styles of forward data-flow analysis implementations.

ForwardAnalysis supports a special mode when no graph is available for traversal (for example, when a CFG is being initialized and constructed, no other graph can be used). In that case, the graph traversal functionality is disabled, and the optimal graph traversal order is not guaranteed. The user can provide a job sorting method to sort the jobs in queue and optimize traversal order.

Feel free to discuss with me (Fish) if you have any suggestions or complaints.

**__init__**(*order_jobs=False*, *allow_merging=False*, *allow_widening=False*, *status_callback=None*, *graph_visitor=None*)

Constructor

**Parameters**

- **order_jobs** (*bool*) – If all jobs should be ordered or not.
- **allow_merging** (*bool*) – If job merging is allowed.

- **allow_widening** (*bool*) – If job widening is allowed.
- **graph_visitor** (Optional[*GraphVisitor*[TypeVar(NodeType)]]) – A graph visitor to provide successors.
- **status_callback** (Callable[[ForwardAnalysis], *Any*] | *None*)

**Returns**
None

**property should_abort**

Should the analysis be terminated. :return: True/False

**property graph: DiGraph**

**property jobs**

**abort()**

Abort the analysis :return: None

**has_job**(*job*)

Checks whether there exists another job which has the same job key. :type job: TypeVar(JobType) :param job: The job to check.

**Return type**
bool

**Returns**
True if there exists another job with the same key, False otherwise.

**Parameters**
**job** (*JobType*)

**downsize()**

**class** angr.analyses.forward_analysis.**FunctionGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**func** (*knowledge.Function*)

**__init__**(*func*, *graph=None*)

**resume_with_new_graph**(*graph*)

We can only reasonably reuse existing results if the node index of the already traversed nodes are the same as the ones from the new graph. Otherwise, we always restart.

**Return type**
bool

**Returns**
True if we are resuming, False if reset() is called.

**Parameters**
**graph** (*DiGraph*)

**class** angr.analyses.forward_analysis.**LoopVisitor**

Bases: *GraphVisitor*

**Parameters**
**loop** (angr.analyses.loopfinder.Loop) – The loop to visit.

__init__(*loop*)

class angr.analyses.forward_analysis.**SingleNodeGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**node** – The single node that should be in the graph.

__init__(*node*)

**node**

**node_returned**

## Submodules

*forward_analysis*
*job_info*
*visitors*

### angr.analyses.forward_analysis.forward_analysis

class angr.analyses.forward_analysis.forward_analysis.**ForwardAnalysis**

Bases: Generic

This is my very first attempt to build a static forward analysis framework that can serve as the base of multiple static analyses in angr, including CFG analysis, VFG analysis, DDG, etc.

In short, ForwardAnalysis performs a forward data-flow analysis by traversing a graph, compute on abstract values, and store results in abstract states. The user can specify what graph to traverse, how a graph should be traversed, how abstract values and abstract states are defined, etc.

ForwardAnalysis has a few options to toggle, making it suitable to be the base class of several different styles of forward data-flow analysis implementations.

ForwardAnalysis supports a special mode when no graph is available for traversal (for example, when a CFG is being initialized and constructed, no other graph can be used). In that case, the graph traversal functionality is disabled, and the optimal graph traversal order is not guaranteed. The user can provide a job sorting method to sort the jobs in queue and optimize traversal order.

Feel free to discuss with me (Fish) if you have any suggestions or complaints.

__init__(*order_jobs=False*, *allow_merging=False*, *allow_widening=False*, *status_callback=None*, *graph_visitor=None*)

Constructor

**Parameters**

- **order_jobs** (*bool*) – If all jobs should be ordered or not.
- **allow_merging** (*bool*) – If job merging is allowed.
- **allow_widening** (*bool*) – If job widening is allowed.
- **graph_visitor** (Optional[*GraphVisitor*[TypeVar(NodeType)]]) – A graph visitor to provide successors.
- **status_callback** (Callable[[ForwardAnalysis, Any] | None)

**Returns**
None

**property should_abort**

Should the analysis be terminated. :return: True/False

**property graph: DiGraph**

**property jobs**

**abort**()

Abort the analysis :return: None

**has_job**(*job*)

Checks whether there exists another job which has the same job key. :type job: TypeVar(JobType) :param job: The job to check.

**Return type**
bool

**Returns**
True if there exists another job with the same key, False otherwise.

**Parameters**
**job** (*JobType*)

**downsize**()

**class** angr.analyses.forward_analysis.forward_analysis.**ForwardAnalysisForDummies**

Bases: *ForwardAnalysis*, Generic

**class** angr.analyses.forward_analysis.forward_analysis.**ForwardAnalysisForClinic**

Bases: *ForwardAnalysisForDummies*[AnalysisState, Block, tuple[int, int | None]], Generic

### angr.analyses.forward_analysis.job_info

**class** angr.analyses.forward_analysis.job_info.**JobInfo**

Bases: Generic

Stores information of each job.

**__init__**(*key*, *job*)

**Parameters**

- **key** (*JobKey*)
- **job** (*JobType*)

**property job: JobType**

Get the latest available job.

**Returns**
The latest available job.

**property merged_jobs**

**property widened_jobs**

**add_job**(*job*, *merged=False*, *widened=False*)

Appended a new job to this JobInfo node. :type job: :param job: The new job to append. :type bool merged: :param bool merged: Whether it is a merged job or not. :type bool widened: :param bool widened: Whether it is a widened job or not.

### angr.analyses.forward_analysis.visitors

**class** angr.analyses.forward_analysis.visitors.**CallGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**callgraph** (*networkx.DiGraph*)

**__init__**(*callgraph*)

**class** angr.analyses.forward_analysis.visitors.**FunctionGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**func** (*knowledge.Function*)

**__init__**(*func*, *graph=None*)

**resume_with_new_graph**(*graph*)

We can only reasonably reuse existing results if the node index of the already traversed nodes are the same as the ones from the new graph. Otherwise, we always restart.

**Return type**
bool

**Returns**
True if we are resuming, False if reset() is called.

**Parameters**
**graph** (*DiGraph*)

**class** angr.analyses.forward_analysis.visitors.**LoopVisitor**

Bases: *GraphVisitor*

**Parameters**
**loop** (angr.analyses.loopfinder.Loop) – The loop to visit.

**__init__**(*loop*)

**class** angr.analyses.forward_analysis.visitors.**SingleNodeGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**node** – The single node that should be in the graph.

**__init__**(*node*)

**node**

**node_returned**

### Submodules


*call_graph*
*function_graph*
*graph*
*loop*
*single_node_graph*


### angr.analyses.forward_analysis.visitors.call_graph

**class** angr.analyses.forward_analysis.visitors.call_graph.**CallGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**callgraph** (*networkx.DiGraph*)

**__init__**(*callgraph*)

### angr.analyses.forward_analysis.visitors.function_graph

**class** angr.analyses.forward_analysis.visitors.function_graph.**FunctionGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**func** (*knowledge.Function*)

**__init__**(*func*, *graph=None*)

**resume_with_new_graph**(*graph*)

We can only reasonably reuse existing results if the node index of the already traversed nodes are the same as the ones from the new graph. Otherwise, we always restart.

**Return type**
bool

**Returns**
True if we are resuming, False if reset() is called.

**Parameters**
**graph** (*DiGraph*)

### angr.analyses.forward_analysis.visitors.graph

**class** angr.analyses.forward_analysis.visitors.graph.**GraphVisitor**

Bases: Generic

A graph visitor takes a node in the graph and returns its successors. Typically, it visits a control flow graph, and returns successors of a CFGNode each time. This is the base class of all graph visitors.

**__init__**()

**successors**(*node*)

Get successors of a node. The node should be in the graph.

**Parameters**
**node** (TypeVar(NodeType)) – The node to work with.

**Returns**
A list of successors.

**Return type**
list[TypeVar(NodeType)]

**predecessors**(*node*)

Get predecessors of a node. The node should be in the graph.

**Parameters**
**node** (TypeVar(NodeType)) – The node to work with.

**Return type**
list[TypeVar(NodeType)]

**Returns**
A list of predecessors.

**sort_nodes**(*nodes=None*)

Get a list of all nodes sorted in an optimal traversal order.

**Parameters**
**nodes** (Collection[TypeVar(NodeType)] | None) – A collection of nodes to sort. If none, all nodes in the graph will be used to sort.

**Return type**
list[TypeVar(NodeType)]

**Returns**
A list of sorted nodes.

**back_edges**()

Get a list of back edges. This function is optional. If not overridden, the traverser cannot achieve an optimal graph traversal order.

**Return type**
list[tuple[TypeVar(NodeType), TypeVar(NodeType)]]

**Returns**
A list of back edges (source -> destination).

**nodes**()

Return an iterator of nodes following an optimal traversal order.

**Return type**
Iterator[TypeVar(NodeType)]

**Returns**

**reset**()

Reset the internal node traversal state. Must be called prior to visiting future nodes.

**Returns**
None

**next_node**()

Get the next node to visit.

**Return type**
Optional[TypeVar(NodeType)]

**Returns**
A node in the graph.

`all_successors`(*node*, *skip_reached_fixedpoint=False*)

Returns all successors to the specific node.

**Parameters**
**node** (TypeVar(NodeType)) – A node in the graph.

**Returns**
A set of nodes that are all successors to the given node.

**Return type**
set[TypeVar(NodeType)]

`revisit_successors`(*node*, *include_self=True*)

Revisit a node in the future. As a result, the successors to this node will be revisited as well.

**Parameters**
**node** (TypeVar(NodeType)) – The node to revisit in the future.

**Return type**
None

**Returns**
None

`revisit_node`(*node*)

Revisit a node in the future. Do not include its successors immediately.

**Parameters**
**node** (TypeVar(NodeType)) – The node to revisit in the future.

**Return type**
None

**Returns**
None

`reached_fixedpoint`(*node*)

Mark a node as reached fixed-point. This node as well as all its successors will not be visited in the future.

**Parameters**
**node** (TypeVar(NodeType)) – The node to mark as reached fixed-point.

**Return type**
None

**Returns**
None

### angr.analyses.forward_analysis.visitors.loop

`class` angr.analyses.forward_analysis.visitors.loop.**LoopVisitor**

Bases: *GraphVisitor*

**Parameters**
**loop** (angr.analyses.loopfinder.Loop) – The loop to visit.

`__init__`(*loop*)

#### angr.analyses.forward_analysis.visitors.single_node_graph

class angr.analyses.forward_analysis.visitors.single_node_graph.**SingleNodeGraphVisitor**

Bases: *GraphVisitor*

**Parameters**
**node** – The single node that should be in the graph.

**__init__**(*node*)

**node**

**node_returned**

### 10.3.28 angr.analyses.identifier

class angr.analyses.identifier.**Identifier**

Bases: *Analysis*

**__init__**(*cfg=None*, *require_predecessors=True*, *only_find=None*)

**run**(*only_find=None*)

**can_call_same_name**(*addr*, *name*)

**get_func_info**(*func*)

**static constrain_all_zero**(*before_state*, *state*, *regs*)

**identify_func**(*function*)

**check_tests**(*cfg_func*, *match_func*)

**map_callsites**()

**do_trace**(*addr_trace*, *reverse_accesses*, *func_info*)

**get_call_args**(*func*, *callsite*)

**static get_reg_name**(*arch*, *reg_offset*)

**Parameters**

- **arch** – the architecture
- **reg_offset** – Tries to find the name of a register given the offset in the registers.

**Returns**
The register name

**find_stack_vars_x86**(*func*)

**static make_initial_state**(*project*, *stack_length*)

**Returns**
an initial state with a symbolic stack and good options for rop

**static make_symbolic_state**(*project*, *reg_list*, *stack_length=80*)

converts an input state into a state with symbolic registers :return: the symbolic state

### Submodules


custom_callable
errors
func
functions
identify
runner


### angr.analyses.identifier.custom_callable

**class** angr.analyses.identifier.custom_callable.**IdentifierCallable**

Bases: object

Callable is a representation of a function in the binary that can be interacted with like a native python function.

If you set perform_merge=True (the default), the result will be returned to you, and you can get the result state with callable.result_state.

Otherwise, you can get the resulting path group (immutable) at callable.result_path_group.

**__init__**(*project*, *addr*, *concrete_only=False*, *perform_merge=False*, *base_state=None*, *toc=None*, *cc=None*, *max_steps=None*)

**Parameters**

- **project** – The project to operate on
- **addr** – The address of the function to use
- **concrete_only** – Throw an exception if the execution splits into multiple paths
- **perform_merge** – Merge all result states into one at the end (only relevant if concrete_only=False)
- **base_state** – The state from which to do these runs
- **toc** – The address of the table of contents for ppc64
- **cc** – The SimCC to use for a calling convention

**set_base_state**(*state*)

Swap out the state you'd like to use to perform the call :type state: :param state: The state to use to perform the call

**get_base_state**(**args*)

**perform_call**(**args*, *prototype=None*)

### angr.analyses.identifier.errors

**exception** angr.analyses.identifier.errors.**IdentifierException**

Bases: *AngrAnalysisError*

**exception** angr.analyses.identifier.errors.**FunctionNotInitialized**

Bases: *AngrAnalysisError*

### angr.analyses.identifier.func

angr.analyses.identifier.func.**rand_str**(*length*, *byte_list=None*)

Generate a random string of *length* characters, drawn from *byte_list* (or all 256 characters if not provided).

**Return type**
str

angr.analyses.identifier.func.**rand_bytes**(*length*, *byte_list=None*)

Generate *length* random bytes, drawn from *byte_list* (or all 256 byte values if not provided).

**Return type**
bytes

**class** angr.analyses.identifier.func.**TestData**

Bases: object

**__init__**(*input_args*, *expected_output_args*, *expected_return_val*, *max_steps*, *preloaded_stdin=None*, *expected_stdout=None*)

**class** angr.analyses.identifier.func.**Func**

Bases: object

**__init__**()

**get_name**()

**num_args**()

**gen_input_output_pair**()

**var_args**()

**can_call_other_funcs**()

**pre_test**(*func*, *runner*)

custom tests run before, return False if it for sure is not the function Use tests here to pick which version of the function :type func: :arg func: the cfg function it will be compared against :type runner: :arg runner: a runner to run the tests with :return: True if we should continue testing

### angr.analyses.identifier.functions

### Submodules

| |
|---|
| *atoi* |
| *based_atoi* |
| *fdprintf* |
| *free* |
| *int2str* |
| *malloc* |
| *memcmp* |
| *memcpy* |
| *memset* |
| *printf* |
| *recv_until* |
| *skip_calloc* |
| *skip_realloc* |

continues on next page

Table 32 – continued from previous page

| |
|---|
| *skip_recv_n* |
| *snprintf* |
| *sprintf* |
| *strcasecmp* |
| *strcmp* |
| *strcpy* |
| *strlen* |
| *strncmp* |
| *strncpy* |
| *strtol* |

### angr.analyses.identifier.functions.atoi

**class** angr.analyses.identifier.functions.atoi.**atoi**

Bases: *Func*

**__init__**()

**num_args**()

**get_name**()

**gen_input_output_pair**()

### angr.analyses.identifier.functions.based_atoi

angr.analyses.identifier.functions.based_atoi.**int2base**(*x*, *base*)

**class** angr.analyses.identifier.functions.based_atoi.**OneTwoOrThree**

Bases: object

**class** angr.analyses.identifier.functions.based_atoi.**based_atoi**

Bases: *Func*

**__init__**()

**num_args**()

**args**()

**get_name**()

**gen_input_output_pair**()

### angr.analyses.identifier.functions.fdprintf

**class** angr.analyses.identifier.functions.fdprintf.**fdprintf**

Bases: *Func*

```
non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t',
'\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14',
'\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e',
'\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':',  ';',  '<', '=', '>',
'?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f',
'\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89',
'\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93',
'\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d',
'\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬',
'\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼',
'½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í',
'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ',
'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']
```

**__init__**()

**num_args**()

**args**()

**get_name**()

**var_args**()

**gen_input_output_pair**()

### angr.analyses.identifier.functions.free

**class** angr.analyses.identifier.functions.free.**free**

Bases: *Func*

**__init__**()

**num_args**()

**get_name**()

**gen_input_output_pair**()

**try_match**(*func*, *identifier*, *runner*)

### angr.analyses.identifier.functions.int2str

**class** angr.analyses.identifier.functions.int2str.**TwoOrThree**

Bases: object

**class** angr.analyses.identifier.functions.int2str.**ThreeOrFour**

Bases: object

**class** angr.analyses.identifier.functions.int2str.**FourOrFive**

Bases: object

class angr.analyses.identifier.functions.int2str.**int2str**

Bases: *Func*

**__init__**()

**num_args**()

**args**()

**get_name**()

**gen_input_output_pair**()

class angr.analyses.identifier.functions.int2str.**int2str_v2**

Bases: *Func*

**__init__**()

**num_args**()

**args**()

**get_name**()

**gen_input_output_pair**()

class angr.analyses.identifier.functions.int2str.**int2str_v3**

Bases: *Func*

**__init__**()

**num_args**()

**args**()

**get_name**()

**gen_input_output_pair**()

class angr.analyses.identifier.functions.int2str.**int2str_v4**

Bases: *Func*

**__init__**()

**num_args**()

**args**()

**get_name**()

**gen_input_output_pair**()

### angr.analyses.identifier.functions.malloc

class angr.analyses.identifier.functions.malloc.**malloc**

Bases: *Func*

**__init__**()

num_args()

get_name()

gen_input_output_pair()

### angr.analyses.identifier.functions.memcmp

class angr.analyses.identifier.functions.memcmp.**memcmp**

Bases: *Func*

__init__()

get_name()

num_args()

args()

gen_input_output_pair()

can_call_other_funcs()

### angr.analyses.identifier.functions.memcpy

class angr.analyses.identifier.functions.memcpy.**memcpy**

Bases: *Func*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t',
'\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14',
'\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e',
'\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>',
'?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f',
'\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89',
'\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93',
'\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d',
'\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬',
'\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼',
'½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í',
'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ',
'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

__init__()

get_name()

num_args()

args()

can_call_other_funcs()

gen_input_output_pair()

### angr.analyses.identifier.functions.memset

class angr.analyses.identifier.functions.memset.**memset**

Bases: *Func*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t', '\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14', '\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e', '\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f', '\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89', '\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93', '\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d', '\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬', '\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼', '½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í', 'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ', 'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï', 'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

**__init__**()

**get_name**()

**num_args**()

**args**()

**can_call_other_funcs**()

**gen_input_output_pair**()

### angr.analyses.identifier.functions.printf

class angr.analyses.identifier.functions.printf.**printf**

Bases: *Func*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t',
'\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14',
'\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e',
'\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>',
'?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f',
'\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89',
'\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93',
'\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d',
'\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬',
'\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼',
'½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í',
'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ',
'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

__init__()

num_args()

args()

get_name()

var_args()

gen_input_output_pair()

### angr.analyses.identifier.functions.recv_until

class angr.analyses.identifier.functions.recv_until.**receive_until_fd**

Bases: *Func*

__init__()

get_name()

num_args()

base_args()

args()

gen_input_output_pair()

do_pretests(*func*, *runner*)

run_test(*func*, *runner*, *test*)

fixup_test(*test*)

class angr.analyses.identifier.functions.recv_until.**receive_until**

Bases: *Func*

__init__()

get_name()

num_args()

base_args()

args()

gen_input_output_pair()

do_pretests(*func*, *runner*)

run_test(*func*, *runner*, *test*)

fixup_test(*test*)

### angr.analyses.identifier.functions.skip_calloc

class angr.analyses.identifier.functions.skip_calloc.**calloc**

Bases: *Func*

__init__()

num_args()

get_name()

args()

gen_input_output_pair()

### angr.analyses.identifier.functions.skip_realloc

class angr.analyses.identifier.functions.skip_realloc.**realloc**

Bases: *Func*

__init__()

num_args()

args()

get_name()

gen_input_output_pair()

### angr.analyses.identifier.functions.skip_recv_n

class angr.analyses.identifier.functions.skip_recv_n.**receive_n4**

Bases: *Func*

__init__()

get_name()

num_args()

args()

gen_input_output_pair()

class angr.analyses.identifier.functions.skip_recv_n.**receive_n3**

Bases: *Func*

__init__()

get_name()

num_args()

args()

gen_input_output_pair()

class angr.analyses.identifier.functions.skip_recv_n.**receive_n2**

Bases: *Func*

__init__()

get_name()

num_args()

args()

gen_input_output_pair()

### angr.analyses.identifier.functions.snprintf

class angr.analyses.identifier.functions.snprintf.**snprintf**

Bases: *Func*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t',
'\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14',
'\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e',
'\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>',
'?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f',
'\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89',
'\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93',
'\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d',
'\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬',
'\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼',
'½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í',
'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ',
'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

__init__()

num_args()

args()

get_name()

var_args()

gen_input_output_pair()

### angr.analyses.identifier.functions.sprintf

class angr.analyses.identifier.functions.sprintf.**sprintf**

Bases: *Func*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t', '\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14', '\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e', '\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f', '\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89', '\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93', '\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d', '\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬', '\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼', '½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í', 'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ', 'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï', 'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

__init__()

num_args()

args()

get_name()

var_args()

gen_input_output_pair()

### angr.analyses.identifier.functions.strcasecmp

class angr.analyses.identifier.functions.strcasecmp.**strcasecmp**

Bases: *strcmp*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t',
'\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14',
'\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e',
'\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>',
'?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f',
'\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89',
'\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93',
'\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d',
'\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬',
'\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼',
'½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í',
'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ',
'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

__init__()

get_name()

num_args()

can_call_other_funcs()

### angr.analyses.identifier.functions.strcmp

class angr.analyses.identifier.functions.strcmp.**strcmp**

Bases: *Func*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t',
'\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14',
'\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e',
'\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.',
'/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>',
'?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_',
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f',
'\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89',
'\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93',
'\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d',
'\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬',
'\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼',
'½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í',
'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ',
'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï',
'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

__init__()

get_name()

num_args()

args()

gen_input_output_pair()

can_call_other_funcs()

### angr.analyses.identifier.functions.strcpy

class angr.analyses.identifier.functions.strcpy.**strcpy**

Bases: *Func*

non_null = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255]

__init__()

get_name()

num_args()

args()

can_call_other_funcs()

gen_input_output_pair()

### angr.analyses.identifier.functions.strlen

class angr.analyses.identifier.functions.strlen.**strlen**

Bases: *Func*

non_null = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255]

num_args()

get_name()

gen_input_output_pair()

### angr.analyses.identifier.functions.strncmp

**class** angr.analyses.identifier.functions.strncmp.**strncmp**

Bases: *Func*

non_null = ['\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08', '\t', '\n', '\x0b', '\x0c', '\r', '\x0e', '\x0f', '\x10', '\x11', '\x12', '\x13', '\x14', '\x15', '\x16', '\x17', '\x18', '\x19', '\x1a', '\x1b', '\x1c', '\x1d', '\x1e', '\x1f', ' ', '!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ':', ';', '<', '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[', '\\', ']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~', '\x7f', '\x80', '\x81', '\x82', '\x83', '\x84', '\x85', '\x86', '\x87', '\x88', '\x89', '\x8a', '\x8b', '\x8c', '\x8d', '\x8e', '\x8f', '\x90', '\x91', '\x92', '\x93', '\x94', '\x95', '\x96', '\x97', '\x98', '\x99', '\x9a', '\x9b', '\x9c', '\x9d', '\x9e', '\x9f', '\xa0', '¡', '¢', '£', '¤', '¥', '¦', '§', '¨', '©', 'ª', '«', '¬', '\xad', '®', '¯', '°', '±', '²', '³', '´', 'µ', '¶', '·', '¸', '¹', 'º', '»', '¼', '½', '¾', '¿', 'À', 'Á', 'Â', 'Ã', 'Ä', 'Å', 'Æ', 'Ç', 'È', 'É', 'Ê', 'Ë', 'Ì', 'Í', 'Î', 'Ï', 'Ð', 'Ñ', 'Ò', 'Ó', 'Ô', 'Õ', 'Ö', '×', 'Ø', 'Ù', 'Ú', 'Û', 'Ü', 'Ý', 'Þ', 'ß', 'à', 'á', 'â', 'ã', 'ä', 'å', 'æ', 'ç', 'è', 'é', 'ê', 'ë', 'ì', 'í', 'î', 'ï', 'ð', 'ñ', 'ò', 'ó', 'ô', 'õ', 'ö', '÷', 'ø', 'ù', 'ú', 'û', 'ü', 'ý', 'þ', 'ÿ']

__init__()

get_name()

num_args()

args()

gen_input_output_pair()

can_call_other_funcs()

### angr.analyses.identifier.functions.strncpy

class angr.analyses.identifier.functions.strncpy.**strncpy**

Bases: *Func*

**non_null** = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255]

**__init__**()

**get_name**()

**num_args**()

**args**()

**can_call_other_funcs**()

**gen_input_output_pair**()

### angr.analyses.identifier.functions.strtol

angr.analyses.identifier.functions.strtol.**int2base**(*x*, *base*)

class angr.analyses.identifier.functions.strtol.**strtol**

Bases: *Func*

**__init__**()

**num_args**()

**args**()

**get_name**()

**gen_input_output_pair**()

### angr.analyses.identifier.identify

class angr.analyses.identifier.identify.**FuncInfo**

Bases: object

**__init__**()

class angr.analyses.identifier.identify.**Identifier**

Bases: *Analysis*

**__init__**(*cfg=None*, *require_predecessors=True*, *only_find=None*)

**run**(*only_find=None*)

**can_call_same_name**(*addr*, *name*)

**get_func_info**(*func*)

**static constrain_all_zero**(*before_state*, *state*, *regs*)

**identify_func**(*function*)

**check_tests**(*cfg_func*, *match_func*)

**map_callsites**()

**do_trace**(*addr_trace*, *reverse_accesses*, *func_info*)

**get_call_args**(*func*, *callsite*)

**static get_reg_name**(*arch*, *reg_offset*)

**Parameters**

- **arch** – the architecture
- **reg_offset** – Tries to find the name of a register given the offset in the registers.

**Returns**
The register name

**find_stack_vars_x86**(*func*)

**static make_initial_state**(*project*, *stack_length*)

**Returns**
an initial state with a symbolic stack and good options for rop

**static make_symbolic_state**(*project*, *reg_list*, *stack_length=80*)

converts an input state into a state with symbolic registers :return: the symbolic state

### angr.analyses.identifier.runner

class angr.analyses.identifier.runner.**Runner**

Bases: object

**__init__**(*project*, *cfg*)

**setup_state**(*function*, *test_data*, *initial_state=None*, *concrete_rand=False*)

**static syscall_hook**(*state*)

**static syscall_hook_concrete_rand**(*state*)

**get_base_call_state**(*function*, *test_data*, *initial_state=None*, *concrete_rand=False*)

**test**(*function*, *test_data*, *concrete_rand=False*, *custom_offs=None*)

**get_out_state**(*function*, *test_data*, *initial_state=None*, *concrete_rand=False*, *custom_offs=None*)

### 10.3.29 angr.analyses.init_finder

**class** angr.analyses.init_finder.**SimEngineInitFinderVEX**

Bases: *SimEngineNostmtVEX*[None, Base | int | None, None]

The VEX engine class for InitFinder.

**__init__**(*project*, *replacements*, *overlay*, *pointers_only=False*)

**static is_concrete**(*expr*)

**Return type**
bool

**class** angr.analyses.init_finder.**InitializationFinder**

Bases: *ForwardAnalysis*, *Analysis*

Finds possible initializations for global data sections and generate an overlay to be used in other analyses later on.

### 10.3.30 angr.analyses.language_detector

**class** angr.analyses.language_detector.**LanguageDetectionConfidenceLevel**

Bases: Enum

Confidence level of language detection results.

**LOW = 'low'**

**MEDIUM = 'medium'**

**HIGH = 'high'**

**class** angr.analyses.language_detector.**LanguageDetectionResult**

Bases: object

Result of language/compiler detection on a binary.

**language:** str = 'unknown'

**compiler:** str | None = None

**compiler_version:** str | None = None

**confidence:** *LanguageDetectionConfidenceLevel* = 'low'

**evidence:** list[str]

**__init__**(*language='unknown'*, *compiler=None*, *compiler_version=None*, *confidence=LanguageDetectionConfidenceLevel.LOW*, *evidence=<factory>*)

**Parameters**

- **language** (*str*)
- **compiler** (*str | None*)
- **compiler_version** (*str | None*)
- **confidence** (LanguageDetectionConfidenceLevel)
- **evidence** (*list[str]*)

**Return type**
None

**class** angr.analyses.language_detector.**LanguageDetector**

Bases: *Analysis*

Detect the original programming language and compiler used to build a binary.

Supports detection of C (gcc, clang, msvc), Rust, Go, and Swift through multiple heuristic layers: DWARF debug info, .comment sections, symbol patterns, section names, and linked library names.

Usage:

```
result = project.analyses.LanguageDetector()
print(result.language)     # "rust"
print(result.compiler)     # "rustc"
print(result.confidence)   # "high"
print(result.evidence)     # ["symbol: __rust_alloc", ...]
```

**__init__**()

**property language:** str

**property compiler:** str | None

**property compiler_version:** str | None

**property confidence:** *LanguageDetectionConfidenceLevel*

**property evidence:** list[str]

### 10.3.31 angr.analyses.loop_analysis

**class** angr.analyses.loop_analysis.**LoopAnalysis**

Bases: *Analysis*

Analyze loop nodes in a structured C code representation and extract relevant information about the loop, including - Loop block addresses - Loop exits - Loop type - Loop condition - Max iterations - Fixed iterations

**__init__**(*cfunc*)

**Parameters**
**cfunc** (CFunction)

#### Submodules

*loop_analysis*

#### angr.analyses.loop_analysis.loop_analysis

**class** angr.analyses.loop_analysis.loop_analysis.**ASTNodeBase**

Bases: object

The base node for AST nodes used in loop analysis.

class angr.analyses.loop_analysis.loop_analysis.**VarNode**

Bases: *ASTNodeBase*

Represents a variable node in the AST; corresponds to a virtual variable in AIL.

**__init__**(*var_ident*)

**Parameters**
**var_ident** (*str*)

class angr.analyses.loop_analysis.loop_analysis.**ConstNode**

Bases: *ASTNodeBase*

Represents a constant value.

**__init__**(*value*)

**Parameters**
**value** (*int*)

class angr.analyses.loop_analysis.loop_analysis.**BinOpNode**

Bases: *ASTNodeBase*

Represents a binary operation between a variable and a constant.

**__init__**(*op*, *lhs*, *rhs*)

**Parameters**

- **op** (*str*)
- **lhs** (VarNode)
- **rhs** (ConstNode)

class angr.analyses.loop_analysis.loop_analysis.**AssignmentNode**

Bases: *ASTNodeBase*

Represents an assignment operation in the AST.

**__init__**(*lhs*, *rhs*)

**Parameters**

- **lhs** (VarNode)
- **rhs** (ASTNodeBase)

class angr.analyses.loop_analysis.loop_analysis.**AssignmentToASTVisitor**

Bases: *CStructuredCodeWalker*

A visitor that converts AIL assignments into AST nodes.

**handle_CAssignment**(*obj*)

**Return type**
*AssignmentNode* | None

**Parameters**
**obj** (CAssignment)

**handle_CVariable**(*obj*)

**Return type**
*VarNode* | None

**Parameters**
**obj** (CVariable)

**handle_CBinaryOp**(*obj*)

**Return type**
*BinOpNode* | None

**Parameters**
**obj** (CBinaryOp)

**handle_CConstant**(*obj*)

**Return type**
*ConstNode* | None

**Parameters**
**obj** (CConstant)

**class** angr.analyses.loop_analysis.loop_analysis.**LoopBodyControlStatementCollector**

Bases: *CStructuredCodeWalker*

A visitor that determines if there are control statements (break, continue, goto) within a loop body.

**__init__**()

**handle_CBreak**(*obj*)

**Parameters**
**obj** (CBreak)

**handle_CContinue**(*obj*)

**handle_CGoto**(*obj*)

**handle_CSwitchCase**(*obj*)

**handle_CForLoop**(*obj*)

**handle_CDoWhileLoop**(*obj*)

**handle_CWhileLoop**(*obj*)

**class** angr.analyses.loop_analysis.loop_analysis.**LoopBodyAssignmentCollector**

Bases: *CStructuredCodeWalker*

A visitor that collects assignments to specific variables within a loop body.

**__init__**(*var_idents*)

**Parameters**
**var_idents** (*set[str]*)

**handle_CAssignment**(*obj*)

**Parameters**
**obj** (CAssignment)

**handle_CIfElse**(*obj*)

**handle_CSwitchCase**(*obj*)

**handle_CForLoop**(*obj*)

**handle_CDoWhileLoop**(*obj*)

**handle_CWhileLoop**(*obj*)

**class** angr.analyses.loop_analysis.loop_analysis.**LoopVisitor**

Bases: *CStructuredCodeWalker*

A visitor that analyzes loop structures in CStructuredCode and collects relevant information for loops.

**__init__**()

**handle_CIfElse**(*obj*)

**handle_CStatements**(*obj*)

**Parameters**
**obj** (CStatements)

**handle_CAssignment**(*obj*)

**Parameters**
**obj** (CAssignment)

**handle_CWhileLoop**(*obj*)

**Parameters**
**obj** (CWhileLoop)

**handle_CDoWhileLoop**(*obj*)

**Parameters**
**obj** (CDoWhileLoop)

**handle_CForLoop**(*obj*)

**Parameters**
**obj** (CForLoop)

**class** angr.analyses.loop_analysis.loop_analysis.**LoopAnalysis**

Bases: *Analysis*

Analyze loop nodes in a structured C code representation and extract relevant information about the loop, including - Loop block addresses - Loop exits - Loop type - Loop condition - Max iterations - Fixed iterations

**__init__**(*cfunc*)

**Parameters**
**cfunc** (CFunction)

### 10.3.32 angr.analyses.loop_unroller

**class** angr.analyses.loop_unroller.**LoopUnroller**

Bases: *Analysis*

Unroll a loop in an AIL graph for a specified number of iterations.

__init__(*graph*, *loop_body*, *unroll_times*, *save_original*, *loop_body_incomplete=False*)

**Parameters**

- **graph** (*DiGraph*)
- **loop_body** (*set[tuple[int, int | None]]*)
- **unroll_times** (*int*)
- **save_original** (*bool*)
- **loop_body_incomplete** (*bool*)

### Submodules

*loop_unroller*

#### angr.analyses.loop_unroller.loop_unroller

**class** angr.analyses.loop_unroller.loop_unroller.**LoopUnroller**

Bases: *Analysis*

Unroll a loop in an AIL graph for a specified number of iterations.

__init__(*graph*, *loop_body*, *unroll_times*, *save_original*, *loop_body_incomplete=False*)

**Parameters**

- **graph** (*DiGraph*)
- **loop_body** (*set[tuple[int, int | None]]*)
- **unroll_times** (*int*)
- **save_original** (*bool*)
- **loop_body_incomplete** (*bool*)

## 10.3.33 angr.analyses.loopfinder

**class** angr.analyses.loopfinder.**Loop**

Bases: object

__init__(*entry*, *entry_edges*, *break_edges*, *continue_edges*, *body_nodes*, *graph*, *subloops*)

**class** angr.analyses.loopfinder.**LoopFinder**

Bases: *Analysis*

Extracts all the loops from all the functions in a binary.

__init__(*functions=None*, *normalize=True*)

## 10.3.34 angr.analyses.outliner

**class** angr.analyses.outliner.**Outliner**

Bases: *Analysis*

Outliner takes a function and some locations and attempts to outline the blocks within these locations into a separate function.

__init__(*func*, *ail_graph*, *src_loc*, *func_entry_loc=None*, *frontier=None*, *vvar_id_start=48879*, *block_addr_start=2864381952*, *min_step=1*, *liveness=None*)

cleanup_callee_graph(*g*, *func*)

Remove all phi assignments whose all source variables are undefined in the graph.

**Parameters**

- **g** (*DiGraph*)
- **func** (Function)

get_interface(*g*, *func*)

Recover the interface from a function AIL graph.

**Parameters**

- **g** (*networkx.DiGraph[Block]*)
- **func** (Function)

**Return type**
list[*VirtualVariable*]

### Submodules

*outliner*

#### angr.analyses.outliner.outliner

class angr.analyses.outliner.outliner.**Outliner**

Bases: *Analysis*

Outliner takes a function and some locations and attempts to outline the blocks within these locations into a separate function.

__init__(*func*, *ail_graph*, *src_loc*, *func_entry_loc=None*, *frontier=None*, *vvar_id_start=48879*, *block_addr_start=2864381952*, *min_step=1*, *liveness=None*)

cleanup_callee_graph(*g*, *func*)

Remove all phi assignments whose all source variables are undefined in the graph.

**Parameters**

- **g** (*DiGraph*)
- **func** (Function)

get_interface(*g*, *func*)

Recover the interface from a function AIL graph.

## 10.3.35 angr.analyses.patchfinder

class angr.analyses.patchfinder.**OverlappingFunctionsAnalysis**

Bases: *Analysis*

Identify functions with interleaved blocks.

__init__()

**overlapping_functions:** dict[int, list[int]]

class angr.analyses.patchfinder.**FunctionAlignmentAnalysis**

Bases: *Analysis*

Determine typical function alignment

**__init__**()

**alignment:** int | None

class angr.analyses.patchfinder.**AtypicallyAlignedFunction**

Bases: object

AtypicallyAlignedFunction(function: 'Function', expected_alignment: 'int')

**function:** *Function*

**expected_alignment:** int

**__init__**(*function*, *expected_alignment*)

**Parameters**

- **function** (Function)
- **expected_alignment** (*int*)

**Return type**
None

class angr.analyses.patchfinder.**PatchedOutFunctionality**

Bases: object

PatchedOutFunctionality(patched_function: 'Function', patched_out_function: 'Function')

**patched_function:** *Function*

**patched_out_function:** *Function*

**__init__**(*patched_function*, *patched_out_function*)

**Parameters**

- **patched_function** (Function)
- **patched_out_function** (Function)

**Return type**
None

class angr.analyses.patchfinder.**PatchFinderAnalysis**

Bases: *Analysis*

Looks for binary patches using some basic heuristics: - Looking for interleaved functions - Looking for unaligned functions

**__init__**()

**atypical_alignments:** list[*AtypicallyAlignedFunction*]

**possibly_patched_out:** list[*PatchedOutFunctionality*]

### 10.3.36 angr.analyses.pathfinder

**exception** angr.analyses.pathfinder.**Unreachable**

Bases: Exception

**class** angr.analyses.pathfinder.**SimStateMarker**

Bases: object

SimStateMarker(addr: 'int', parent: 'SimStateMarker | None' = None, banned: 'bool' = False, misses: 'int' = 0)

**addr:** *int*

**parent:** *SimStateMarker* | None = None

**banned:** bool = False

**misses:** int = 0

**__init__**(*addr*, *parent=None*, *banned=False*, *misses=0*)

**Parameters**

- **addr** (*int*)
- **parent** (SimStateMarker | *None*)
- **banned** (*bool*)
- **misses** (*int*)

**Return type**
None

**class** angr.analyses.pathfinder.**SuccessorsKind**

Bases: Enum

**SAT** = 1

**UNSAT** = 2

**MISSING** = 3

**class** angr.analyses.pathfinder.**TestPathReport**

Bases: object

TestPathReport(path_markers: 'dict[int, SimStateMarker]', termination: 'SuccessorsKind')

**path_markers:** dict[int, *SimStateMarker*]

**termination:** *SuccessorsKind*

**__init__**(*path_markers*, *termination*)

**Parameters**

- **path_markers** (*dict[int, SimStateMarker]*)
- **termination** (SuccessorsKind)

**Return type**
None

angr.analyses.pathfinder.**nilref**()

class angr.analyses.pathfinder.**Pathfinder**

Bases: *Analysis*

**__init__**(*start_state*, *goal_addr*, *cfg*, *cache_size=10000*)

**Parameters**

- **start_state** (SimState)
- **goal_addr** (*int*)
- **cfg** (CFGModel)

**goal_state:** *SimState* | None

**transition_cache:** DiGraph[*SimStateMarker*]

**base_heuristic:** dict[int, int]

**cache_state**(*state*)

**Parameters**
**state** (SimState)

**marker_to_state**(*marker*)

**Return type**
*SimState* | None

**Parameters**
**marker** (SimStateMarker)

**analyze**()

**Return type**
bool

**find_best_hypothesis_path**()

**Return type**
tuple[int, ...]

**diagnose_unsat**(*state*)

**Parameters**
**state** (SimState)

**test_path**(*bbl_addr_trace*)

**Return type**
*TestPathReport*

**Parameters**
**bbl_addr_trace** (*tuple[int, ...]*)

angr.analyses.pathfinder.**find_successor**(*successors*, *target_addr*)

**Return type**
tuple[*SimState* | None, *SuccessorsKind*]

**Parameters**

- **successors** (SimSuccessors)
- **target_addr** (*int*)

### 10.3.37 angr.analyses.propagator

**class** angr.analyses.propagator.**PropagatorAnalysis**

Bases: *ForwardAnalysis*, *Analysis*

PropagatorAnalysis implements copy propagation. It propagates values (either constant values or variables) and expressions inside a block or across a function.

PropagatorAnalysis only supports VEX. For AIL, please use SPropagator.

PropagatorAnalysis performs certain arithmetic operations between constants, including but are not limited to:

- addition
- subtraction
- multiplication
- division
- xor

It also performs the following memory operations:

- Loading values from a known address
- Writing values to a stack variable

**property prop_key:** `tuple[str | None, str, int, bool, bool, bool]`

Gets a key that represents the function and the "flavor" of the propagation result.

**property replacements**

#### Submodules

| |
|---|
| *engine_base* |
| *engine_vex* |
| *propagator* |
| *top_checker_mixin* |
| *values* |
| *vex_vars* |

#### angr.analyses.propagator.engine_base

**class** angr.analyses.propagator.engine_base.**SimEnginePropagatorBaseMixin**

Bases: *SimEngineLight*, Generic

The base class for the propagator VEX engine.

**__init__**(*project*, *stack_pointer_tracker=None*, *propagate_tmps=True*, *reaching_definitions=None*, *bp_as_gpr=False*)

**Parameters**

- **project** (Project)
- **reaching_definitions** (ReachingDefinitionsModel | *None*)
- **bp_as_gpr** (*bool*)

**process**(*state*, *, *block=None*, *base_state=None*, *load_callback=None*, ***kwargs*)

**Return type**
TypeVar(StateType)

**Parameters**

- **state** (StateType)
- **block** (*BlockType | None*)

### angr.analyses.propagator.engine_vex

**class** angr.analyses.propagator.engine_vex.**SimEnginePropagatorVEX**

Bases: *ClaripyDataVEXEngineMixin*[*PropagatorVEXState*, BV, *PropagatorVEXState*, None], *SimEnginePropagatorBaseMixin*[*PropagatorVEXState*, BV, *Block*], *SimEngineNostmtVEX*[*PropagatorVEXState*, BV, *PropagatorVEXState*]

### angr.analyses.propagator.propagator

**class** angr.analyses.propagator.propagator.**PropagatorAnalysis**

Bases: *ForwardAnalysis*, *Analysis*

PropagatorAnalysis implements copy propagation. It propagates values (either constant values or variables) and expressions inside a block or across a function.

PropagatorAnalysis only supports VEX. For AIL, please use SPropagator.

PropagatorAnalysis performs certain arithmetic operations between constants, including but are not limited to:

- addition
- subtraction
- multiplication
- division
- xor

It also performs the following memory operations:

- Loading values from a known address
- Writing values to a stack variable

**property prop_key:** tuple[str | None, str, int, bool, bool, bool]

Gets a key that represents the function and the "flavor" of the propagation result.

**property replacements**

### angr.analyses.propagator.top_checker_mixin

**class** angr.analyses.propagator.top_checker_mixin.**ClaripyDataEngineMixin**

Bases: *SimEngineLight*[StateType, DataType_co | BV, BlockType, ResultType], Generic

**class** angr.analyses.propagator.top_checker_mixin.**ClaripyDataVEXEngineMixin**

Bases: *ClaripyDataEngineMixin*[StateType, DataType_co, *Block*, ResultType], *SimEngineLightVEX*[StateType, DataType_co | BV, ResultType, StmtDataType], Generic

#### angr.analyses.propagator.values

class angr.analyses.propagator.values.**Top**

Bases: object

**__init__**(*size*)

**size**

property **bits**

class angr.analyses.propagator.values.**Bottom**

Bases: object

#### angr.analyses.propagator.vex_vars

class angr.analyses.propagator.vex_vars.**VEXVariable**

Bases: object

class angr.analyses.propagator.vex_vars.**VEXMemVar**

Bases: object

**__init__**(*addr*, *size*)

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**addr**

**size**

class angr.analyses.propagator.vex_vars.**VEXReg**

Bases: *VEXVariable*

**__init__**(*offset*, *size*)

**offset**

**size**

class angr.analyses.propagator.vex_vars.**VEXTmp**

Bases: *VEXVariable*

**__init__**(*tmp*)

**tmp**

### 10.3.38 angr.analyses.proximity_graph

class angr.analyses.proximity_graph.**ProxiNodeTypes**

Bases: object

Node Type Enums

**Empty = 0**

**String = 1**

**Function = 2**

**FunctionCall = 3**

**Integer = 4**

**Unknown = 5**

**Variable = 6**

**class** angr.analyses.proximity_graph.**BaseProxiNode**

Bases: object

Base class for all nodes in a proximity graph.

**__init__**(*type_*, *ref_at=None*)

**Parameters**

- **type_** (*int*)
- **ref_at** (*set[int] | None*)

**class** angr.analyses.proximity_graph.**FunctionProxiNode**

Bases: *BaseProxiNode*

Proximity node showing current and expanded function calls in graph.

**__init__**(*func*, *ref_at=None*)

**Parameters**

- **func** (FuncNode)
- **ref_at** (*set[int] | None*)

**class** angr.analyses.proximity_graph.**VariableProxiNode**

Bases: *BaseProxiNode*

Variable arg node

**__init__**(*addr*, *name*, *ref_at=None*)

**Parameters**
**ref_at** (*set[int] | None*)

**class** angr.analyses.proximity_graph.**StringProxiNode**

Bases: *BaseProxiNode*

String arg node

**__init__**(*addr*, *content*, *ref_at=None*)

**Parameters**
**ref_at** (*set[int] | None*)

**class** angr.analyses.proximity_graph.**CallProxiNode**

Bases: *BaseProxiNode*

Call node

__init__(*callee*, *ref_at=None*, *args=None*)

**Parameters**

- **callee** (FuncNode)
- **ref_at** (*set[int] | None*)
- **args** (*tuple[BaseProxiNode] | None*)

class angr.analyses.proximity_graph.**IntegerProxiNode**

Bases: *BaseProxiNode*

Int arg node

__init__(*value*, *ref_at=None*)

**Parameters**

- **value** (*int*)
- **ref_at** (*set[int] | None*)

class angr.analyses.proximity_graph.**UnknownProxiNode**

Bases: *BaseProxiNode*

Unknown arg node

__init__(*dummy_value*)

**Parameters**
**dummy_value** (*str*)

class angr.analyses.proximity_graph.**ProximityGraphAnalysis**

Bases: *Analysis*

Generate a proximity graph.

__init__(*func*, *cfg_model*, *xrefs*, *decompilation=None*, *expand_funcs=None*)

**Parameters**

- **func** (Function)
- **cfg_model** (CFGModel)
- **xrefs** (XRefManager)
- **decompilation** (Decompiler | *None*)
- **expand_funcs** (*set[int] | None*)

**graph: DiGraph | None**

### 10.3.39 angr.analyses.purity

class angr.analyses.purity.**AILPurityAnalysis**

Bases: *Analysis*, *ForwardAnalysisForClinic*[*StateType*]

Determined how data sources are used in a given function.

angr.analyses.purity.**AILPurityDataSource**

alias of *DataSource*

angr.analyses.purity.**AILPurityDataUsage**

alias of *DataUsage*

angr.analyses.purity.**AILPurityResultType**

alias of *ResultType*

### Submodules

*analysis*
*engine*

#### angr.analyses.purity.analysis

**class** angr.analyses.purity.analysis.**AILPurityAnalysis**

Bases: *Analysis*, *ForwardAnalysisForClinic*[*StateType*]

Determined how data sources are used in a given function.

#### angr.analyses.purity.engine

**class** angr.analyses.purity.engine.**DataUsage**

Bases: object

Facts about a given data source

**ptr_load:** bool = False

**ptr_store:** bool = False

**__init__**(*ptr_load=False*, *ptr_store=False*)

**Parameters**

- **ptr_load** (*bool*)
- **ptr_store** (*bool*)

**Return type**
None

**class** angr.analyses.purity.engine.**DataSource**

Bases: object

Descriptor of a given data source: where did this value come from?

**constant_value:** int | None = None

**function_arg:** int | None = None

**callee_return:** *Function* | str | None = None

**reference_to:** int | None = None

**property callee_return_name:** str | None

__init__(*constant_value=None*, *function_arg=None*, *callee_return=None*, *reference_to=None*)

**Parameters**

- **constant_value** (*int | None*)
- **function_arg** (*int | None*)
- **callee_return** (Function | *str | None*)
- **reference_to** (*int | None*)

**Return type**
None

class angr.analyses.purity.engine.**StateType**

Bases: object

Internal state for purity analysis

**addr:** tuple[int, int | None]

**vars:** MutableMapping[int, *DataType_co*]

**copy**()

__init__(*addr*, *vars*)

**Parameters**

- **addr** (*tuple[int, int | None]*)
- **vars** (*MutableMapping[int,* DataType_co*]*)

**Return type**
None

class angr.analyses.purity.engine.**ResultType**

Bases: object

Result of purity analysis - how was each data source used and what function calls arguments were passed?

**uses:** MutableMapping[*DataSource*, *DataUsage*]

**call_args:** MutableMapping[tuple[int, int | None, int, *Function* | str | None, int], *DataType_co*]

**ret_vals:** MutableMapping[int, *DataType_co*]

**other_storage:** MutableMapping[*DataSource*, *DataType_co*]

**update**(*other*)

**Parameters**
**other** (*Self*)

**is_pure**(*pure_functions=None*, *allow_read_arguments=True*, *allow_write_arguments=False*, *allow_read_globals=True*, *allow_write_globals=False*)

**Return type**
bool

**Parameters**

- **pure_functions** (*set[str] | None*)

- **allow_read_arguments** (*bool*)
- **allow_write_arguments** (*bool*)
- **allow_read_globals** (*bool*)
- **allow_write_globals** (*bool*)

**__init__**(*uses=<factory>*, *call_args=<factory>*, *ret_vals=<factory>*, *other_storage=<factory>*)

**Parameters**

- **uses** (*MutableMapping[DataSource, DataUsage]*)
- **call_args** (*MutableMapping[tuple[int, int | None, int, Function | str | None, int], DataType_co]*)
- **ret_vals** (*MutableMapping[int, DataType_co]*)
- **other_storage** (*MutableMapping[DataSource, DataType_co]*)

**Return type**
None

**type** angr.analyses.purity.engine.**DataType_co** = **frozenset**[*DataSource*]

**type** angr.analyses.purity.engine.**StmtDataType** = **None**

**class** angr.analyses.purity.engine.**PurityEngineAIL**

Bases: *SimEngineLightAIL*[*StateType*, *DataType_co*, *StmtDataType*, *ResultType*]

Core of this analysis: THIS data source has THESE uses.

A use is arithmetic, store, or load.

**__init__**(*project*, *clinic*, *recurse*)

**Parameters**

- **project** (*Project*)
- **clinic** (*Clinic*)
- **recurse** (*Callable[[Function], ResultType]*)

**initial_state**(*node*)

**Parameters**
**node** (*Block*)

**process**(*state*, *, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
*ResultType*

**Parameters**

- **state** (*StateType*)
- **block** (*Block | None*)
- **whitelist** (*set[int] | None*)

### 10.3.40 angr.analyses.reaching_definitions

**class** angr.analyses.reaching_definitions.**Atom**

Bases: object

This class represents a data storage location manipulated by IR instructions.

It could either be a Tmp (temporary variable), a Register, a MemoryLocation.

**__init__**(*size*)

**Parameters**
**size** – The size of the atom in bytes

**size**

**property bits: int**

**static from_ail_expr**(*expr*, *arch*, *full_reg=False*)

**Return type**
*Register*

**Parameters**

- **expr** (Expression)
- **arch** (*Arch*)
- **full_reg** (*bool*)

**static from_argument**(*argument*, *arch*, *full_reg=False*, *sp=None*)

Instantiate an *Atom* from a given argument.

**Parameters**

- **argument** (*SimFunctionArgument*) – The argument to create a new atom from.
- **arch** (Arch) – The argument representing archinfo architecture for argument.
- **full_reg** – Whether to return an atom indicating the entire register if the argument only specifies a slice of the register.
- **sp** (int | None) – The current stack offset. Optional. Only used when argument is a SimStackArg.

**Return type**
*Register* | *MemoryLocation*

**static reg**(*thing*, *size=None*, *arch=None*)

Create a Register atom.

**Parameters**

- **thing** (str | RegisterOffset) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").
- **size** (int | None) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Arch | None) – The architecture. Must be provided when creating the atom using a register name.

**Return type**
*Register*

**Returns**
The Register Atom object.

**static register**(*thing*, *size=None*, *arch=None*)

Create a Register atom.

**Parameters**

- **thing** (str | RegisterOffset) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").
- **size** (int | None) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Arch | None) – The architecture. Must be provided when creating the atom using a register name.

**Return type**
*Register*

**Returns**
The Register Atom object.

**static mem**(*addr*, *size*, *endness=None*)

Create a MemoryLocation atom,

**Parameters**

- **addr** (*SpOffset* | *HeapAddress* | int) – The memory location. Can be an SpOffset for stack variables, an int for global memory variables, or a HeapAddress for items on the heap.
- **size** (int) – Size of the atom.
- **endness** (Endness | None) – Optional, either "Iend_LE" or "Iend_BE".

**Return type**
*MemoryLocation*

**Returns**
The MemoryLocation Atom object.

**static memory**(*addr*, *size*, *endness=None*)

Create a MemoryLocation atom,

**Parameters**

- **addr** (*SpOffset* | *HeapAddress* | int) – The memory location. Can be an SpOffset for stack variables, an int for global memory variables, or a HeapAddress for items on the heap.
- **size** (int) – Size of the atom.
- **endness** (Endness | None) – Optional, either "Iend_LE" or "Iend_BE".

**Return type**
*MemoryLocation*

**Returns**
The MemoryLocation Atom object.

**class** angr.analyses.reaching_definitions.**AtomKind**

Bases: Enum

An enum indicating the class of an atom

**REGISTER = 1**

**MEMORY = 2**

**TMP = 3**

**GUARD = 4**

**CONSTANT = 5**

**class** angr.analyses.reaching_definitions.**ConstantSrc**

Bases: *Atom*

Represents a constant.

**value:** int

**class** angr.analyses.reaching_definitions.**Definition**

Bases: Generic

An atom definition.

**Variables**

- **atom** – The atom being defined.
- **codeloc** – Where this definition is created in the original binary code.
- **dummy** – Tell whether the definition should be considered dummy or not. During simplification by AILment, definitions marked as dummy will not be removed.
- **tags** – A set of tags containing information about the definition gathered during analyses.

**__init__**(*atom*, *codeloc*, *dummy=False*, *tags=None*)

**Parameters**

- **atom** (*A*)
- **codeloc** (*CodeLoc*)
- **dummy** (*bool*)
- **tags** (*set[Tag] | None*)

**atom**

**codeloc**

**dummy**

**tags**

**property offset:** int

**property size:** int

**matches**(***kwargs*)

Return whether this definition has certain characteristics.

**Return type**
bool

class angr.analyses.reaching_definitions.**FunctionCallData**

Bases: object

A bundle of intermediate data used when computing the sum effect of a function during ReachingDefinitions-Analysis.

RDA engine contract:

- Construct one of these before calling *FunctionHandler.handle_function*. Fill it with as many fields as you can realistically provide without duplicating effort.
- Provide *callsite_codeloc* as either the call statement (AIL) or the default exit of the default statement of the calling block (VEX)
- Provide *function_codeloc* as the callee address with *stmt_idx=0* `.

Function handler contract:

- If redefine_locals is unset, do not adjust any artifacts of the function call abstraction, such as the stack pointer, the caller saved registers, etc.
- If caller_will_handle_single_ret is set, and there is a single entry in *ret_atoms*, do not apply to the state effects modifying this atom. Instead, set *ret_values* and *ret_values_deps* to the values and deps which are used constructing these values.

**callsite_codeloc:** *CodeLocation*

**function_codeloc:** *CodeLocation*

**address_multi:** *MultiValues*[BV | FP] | None

**address:** int | None = None

**symbol:** Symbol | None = None

**function:** *Function* | None = None

**name:** str | None = None

**cc:** *SimCC* | None = None

**prototype:** *SimTypeFunction* | None = None

**args_atoms:** list[set[*Atom*]] | None = None

**args_values:** list[*MultiValues*[BV | FP]] | None = None

**ret_atoms:** set[*Atom*] | None = None

**redefine_locals:** bool = True

**visited_blocks:** set[int] | None = None

**effects:** list[*FunctionEffect*]

**ret_values:** *MultiValues*[BV | FP] | None = None

**ret_values_deps:** set[*Definition*] | None = None

**caller_will_handle_single_ret:** bool = False

**guessed_cc:** bool = False

**guessed_prototype:** bool = False

**retaddr_popped:** bool = False

**has_clobbered**(*dest*)

Determines whether the given atom already has effects applied

**Return type**
bool

**Parameters**
**dest** (Atom)

**depends**(*dest*, **sources*, *value=None*, *apply_at_callsite=False*, *tags=None*)

Mark a single effect of the current function, including the atom being modified, the input atoms on which that output atom depends, the precise (or imprecise!) value to store, and whether the effect should be applied during the function or afterwards, at the callsite.

The tags are used to annotate the Definition of the Atom that will be created, when the function effects are applied to the state.

The atom being modified may be None to mark uses of the source atoms which do not have any explicit sinks.

**Parameters**

- **dest** (Atom | *Iterable[Atom]* | *None*)
- **sources** (Atom | *Iterable[Atom]*)
- **value** (MultiValues | *BV* | bytes | *int* | *None*)
- **apply_at_callsite** (*bool*)
- **tags** (*set[Tag]* | *None*)

**reset_prototype**(*prototype*, *state*, *soft_reset=False*)

**Return type**
set[*Atom*]

**Parameters**

- **prototype** (SimTypeFunction)
- **state** (ReachingDefinitionsState)
- **soft_reset** (*bool*)

**__init__**(*callsite_codeloc*, *function_codeloc*, *address_multi*, *address=None*, *symbol=None*, *function=None*, *name=None*, *cc=None*, *prototype=None*, *args_atoms=None*, *args_values=None*, *ret_atoms=None*, *redefine_locals=True*, *visited_blocks=None*, *effects=<factory>*, *ret_values=None*, *ret_values_deps=None*, *caller_will_handle_single_ret=False*, *guessed_cc=False*, *guessed_prototype=False*, *retaddr_popped=False*)

**Parameters**

- **callsite_codeloc** (CodeLocation)
- **function_codeloc** (CodeLocation)
- **address_multi** (MultiValues[*BV* | *FP*] | *None*)
- **address** (*int* | *None*)

- **symbol** (*Symbol | None*)
- **function** (*Function | None*)
- **name** (*str | None*)
- **cc** (*SimCC | None*)
- **prototype** (*SimTypeFunction | None*)
- **args_atoms** (*list[set[Atom]] | None*)
- **args_values** (*list[MultiValues[BV | FP]] | None*)
- **ret_atoms** (*set[Atom] | None*)
- **redefine_locals** (*bool*)
- **visited_blocks** (*set[int] | None*)
- **effects** (*list[FunctionEffect]*)
- **ret_values** (*MultiValues[BV | FP] | None*)
- **ret_values_deps** (*set[Definition] | None*)
- **caller_will_handle_single_ret** (*bool*)
- **guessed_cc** (*bool*)
- **guessed_prototype** (*bool*)
- **retaddr_popped** (*bool*)

**Return type**
None

**class** angr.analyses.reaching_definitions.**FunctionCallRelationships**

Bases: object

Produced by the function handler, provides associated callsite info and function input/output definitions.

**callsite:** *CodeLocation*

**target:** int | None

**args_defns:** list[set[*Definition*]]

**other_input_defns:** set[*Definition*]

**ret_defns:** set[*Definition*]

**other_output_defns:** set[*Definition*]

**__init__**(*callsite*, *target*, *args_defns*, *other_input_defns*, *ret_defns*, *other_output_defns*)

**Parameters**

- **callsite** (CodeLocation)
- **target** (*int | None*)
- **args_defns** (*list[set[Definition]]*)
- **other_input_defns** (*set[Definition]*)
- **ret_defns** (*set[Definition]*)
- **other_output_defns** (*set[Definition]*)

**Return type**
None

class angr.analyses.reaching_definitions.**FunctionHandler**

Bases: object

A mechanism for summarizing a function call's effect on a program for ReachingDefinitionsAnalysis.

**__init__**(*interfunction_level=0*, *extra_impls=None*)

**Parameters**

- **interfunction_level** (int) – Maximum depth in to continue local function exploration
- **extra_impls** (Iterable[type[*FunctionHandler*]] | None) – FunctionHandler classes to implement beyond what's implemented in function_handler_library

**interfunction_level**: int

**hook**(*analysis*)

Attach this instance of the function handler to an instance of RDA.

**Return type**
*FunctionHandler*

**Parameters**
**analysis** (ReachingDefinitionsAnalysis)

**make_function_codeloc**(*target*, *callsite*, *callsite_func_addr*)

The RDA engine will call this function to transform a callsite CodeLocation into a callee CodeLocation.

**Parameters**

- **target** (*int* | MultiValues | *None*)
- **callsite** (CodeLocation)
- **callsite_func_addr** (*int* | *None*)

**handle_function**(*state*, *data*)

The main entry point for the function handler. Called with a RDA state and a FunctionCallData, it is expected to update the state and the data as per the contracts described on FunctionCallData.

You can override this method to take full control over how data is processed, or override any of the following to use the higher-level interface (data.depends()):

- *handle_impl_<function name>* - used for *<function name>*.
- *handle_local_function* - used for any function (excluding plt stubs) whose address is inside the main binary.
- *handle_external_function* - used for any function or plt stub whose address is outside the main binary.
- *handle_indirect_function* - used for any function whose target cannot be resolved.
- *handle_generic_function* - used as a default if none of the above are overridden.

Each of them take the same signature as *handle_function*.

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_generic_function**(*state*, *data*)

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_indirect_function**(*state*, *data*)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_local_function**(*state*, *data*)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_external_function**(*state*, *data*)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**recurse_analysis**(*state*, *data*)

Precondition: `data.function` MUST NOT BE NONE in order to call this method.

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**static c_args_as_atoms**(*state*, *cc*, *prototype*)

**Return type**
list[set[*Atom*]]

**Parameters**

- **state** (ReachingDefinitionsState)
- **cc** (SimCC)
- **prototype** (SimTypeFunction)

**static c_return_as_atoms**(*state*, *cc*, *prototype*)

**Return type**
set[*Atom*]

**Parameters**

- **state** (ReachingDefinitionsState)
- **cc** (SimCC)
- **prototype** (SimTypeFunction)

**static caller_saved_regs_as_atoms**(*state*, *cc*)

**Return type**
set[*Register*]

**Parameters**

- **state** (ReachingDefinitionsState)
- **cc** (SimCC)

**static stack_pointer_as_atom**(*state*)

**Return type**
*Register*

**class** angr.analyses.reaching_definitions.**GuardUse**

Bases: *Atom*

Implements a guard use.

**target**

**class** angr.analyses.reaching_definitions.**LiveDefinitions**

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

**INITIAL_SP_16BIT = 32512**

**INITIAL_SP_32BIT = 2147418112**

**INITIAL_SP_64BIT = 140737488289792**

**__init__**(*arch*, *track_tmps=False*, *canonical_size=8*, *registers=None*, *stack=None*, *memory=None*, *heap=None*, *tmps=None*, *others=None*, *register_uses=None*, *stack_uses=None*, *heap_uses=None*, *memory_uses=None*, *tmp_uses=None*, *other_uses=None*, *element_limit=5*, *merge_into_tops=True*)

**Parameters**

- **arch** (*Arch*)
- **track_tmps** (*bool*)
- **merge_into_tops** (*bool*)

**project: *Project* | None**

**arch**

**track_tmps**

**registers:** *MultiValuedMemory*

**stack:** *MultiValuedMemory*

**memory:** *MultiValuedMemory*

**heap:** *MultiValuedMemory*

**tmps:** dict[int, set[*Definition*]]

**others:** dict[*Atom*, *MultiValues*]

**register_uses**

**stack_uses**

**heap_uses**

**memory_uses**

**tmp_uses:** dict[int, set[*CodeLocation*]]

**other_uses**

**uses_by_codeloc:** dict[*CodeLocation*, set[*Definition*]]

**copy**(*discard_tmpdefs=False*)

**Return type**
*LiveDefinitions*

**reset_uses**()

**static top**(*bits*)

Get a TOP value.

**Parameters**
**bits** (int) – Width of the TOP value (in bits).

**Returns**
The TOP value.

**static is_top**(*expr*)

Check if the given expression is a TOP value.

**Parameters**
**expr** – The given expression.

**Return type**
bool

**Returns**
True if the expression is TOP, False otherwise.

**stack_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int*)

static **is_stack_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

static **get_stack_offset**(*addr*, *had_stack_base=False*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

static **annotate_with_def**(*symvar*, *definition*)

**Parameters**

- **symvar** (TypeVar(MVType, bound= BV | FP))
- **definition** (*Definition*)

**Return type**
TypeVar(MVType, bound= BV | FP)

**Returns**

static **extract_defs**(*symvar*)

**Return type**
Generator[*Definition*]

**Parameters**
**symvar** (*Base*)

static **extract_defs_from_annotations**(*annos*)

**Return type**
set[*Definition*]

**Parameters**
**annos** (*Iterable[Annotation]*)

static **extract_defs_from_mv**(*mv*)

**Return type**
Generator[*Definition*]

**Parameters**
**mv** (MultiValues)

**get_sp**()
Return the concrete value contained by the stack pointer.

**Return type**
int

**get_sp_offset**()
Return the offset of the stack pointer.

**Return type**
int | None

**get_stack_address**(*offset*)

**Return type**
int | None

**Parameters**
**offset** (*Base*)

**stack_offset_to_stack_addr**(*offset*)

**Return type**
int

**merge**(**others*)

**Return type**
tuple[*LiveDefinitions*, bool]

**Parameters**
**others** (LiveDefinitions)

**compare**(*other*)

**Return type**
bool

**Parameters**
**other** (LiveDefinitions)

**kill_definitions**(*atom*)

Overwrite existing definitions w.r.t 'atom' with a dummy definition instance. A dummy definition will not be removed during simplification.

**Parameters**
**atom** (*Atom*)

**Return type**
None

**Returns**
None

**kill_and_add_definition**(*atom*, *code_loc*, *data*, *dummy=False*, *tags=None*, *endness=None*, *annotated=False*)

**Return type**
*MultiValues* | None

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **data** (MultiValues)
- **tags** (*set[Tag] | None*)

**add_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_use_by_def**(*definition*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **definition** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any*)

**get_definitions**(*thing*)

**Parameters**
**thing** (Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]] | MultiValues)

**Return type**
set[*Definition*[*Atom*]]

**get_tmp_definitions**(*tmp_idx*)

**Return type**
set[*Definition*]

**Parameters**
**tmp_idx** (*int*)

**get_register_definitions**(*reg_offset*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)

**get_stack_values**(*stack_offset*, *size*, *endness*)

**Return type**
*MultiValues* | None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **endness** (*str*)

**get_stack_definitions**(*stack_offset*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)

**get_heap_definitions**(*heap_addr*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **heap_addr** (*int*)
- **size** (*int*)

**get_memory_definitions**(*addr*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**get_values**(*spec*, *endness=None*)

**Parameters**

- **spec** (*A | Definition[A] | Iterable[A] | Iterable[Definition[A]]*)
- **endness** (*archinfo.Endness | None*)

**Return type**
*MultiValues* | None

**get_one_value**(*spec*, *strip_annotations=False*)

**Parameters**

- **spec** (*A | Definition[A] | Iterable[A] | Iterable[Definition[A]]*)
- **strip_annotations** (*bool*)

**Return type**
claripy.ast.bv.BV | None

**get_concrete_value**(*spec*, *cast_to=<class 'int'>*)

**Overloads**

- **self**, **spec** (A | Definition[A] | Iterable[A] | Iterable[Definition[A]]), **cast_to** (type[int]) → int | None
- **self**, **spec** (A | Definition[A] | Iterable[A] | Iterable[Definition[A]]), **cast_to** (type[bytes]) → bytes | None

**Parameters**

- **spec** (*A | Definition[A] | Iterable[A] | Iterable[Definition[A]]*)
- **cast_to** (*type[int | bytes]*)

**Return type**
int | *bytes* | None

add_register_use(*reg_offset*, *size*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

add_register_use_by_def(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

add_stack_use(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

add_stack_use_by_def(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

add_heap_use(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_heap_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_memory_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_memory_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_tmp_use**(*atom*, *code_loc*)

**Return type**
None

**Parameters**

- **atom** (Tmp)
- **code_loc** (CodeLocation)

**add_tmp_use_by_def**(*def_*, *code_loc*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)

**deref**(*pointer*, *size*, *endness=Endness.BE*)

**Overloads**

- **self**, **pointer** (MultiValues[claripy.ast.BV] | Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]), **size** (int | DerefSize), **endness** (archinfo.Endness) → set[MemoryLocation]

- **self**, **pointer** (int | claripy.ast.BV | HeapAddress | SpOffset), **size** (int | DerefSize), **endness** (archinfo.Endness) → MemoryLocation | None

**static is_heap_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**static get_heap_offset**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**heap_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int | HeapAddress*)

**class** angr.analyses.reaching_definitions.**MemoryLocation**

Bases: *Atom*

Represents a memory slice.

It is characterized by its address and its size.

**__init__**(*addr*, *size*, *endness=None*)

**Parameters**

- **addr** (*SpOffset* | *HeapAddress* | int) – The address of the beginning memory location slice.
- **size** (int) – The size of the represented memory location, in bytes.
- **endness** (*Endness | None*)

**addr: *SpOffset* | int | BV**

**endness**

**property is_on_stack: bool**

True if this memory location is located on the stack.

**property symbolic: bool**

**class** angr.analyses.reaching_definitions.**ObservationPointType**

Bases: IntEnum

Enum to replace the previously generic constants This makes it possible to annotate where they are expected by typing something as ObservationPointType instead of Literal[0,1]

**OP_BEFORE = 0**

**OP_AFTER = 1**

__new__(*value*)

class angr.analyses.reaching_definitions.**ReachingDefinitionsAnalysis**

Bases: *ForwardAnalysis*[*ReachingDefinitionsState*, NodeType, object, object, object], *Analysis*

ReachingDefinitionsAnalysis is a text-book implementation of a static data-flow analysis that works on either a function or a block. It supports both VEX and AIL. By registering observers to observation points, users may use this analysis to generate use-def chains, def-use chains, and reaching definitions, and perform other traditional data-flow analyses such as liveness analysis.

- I've always wanted to find a better name for this analysis. Now I gave up and decided to live with this name for the foreseeable future (until a better name is proposed by someone else).
- Aliasing is definitely a problem, and I forgot how aliasing is resolved in this implementation. I'll leave this as a post-graduation TODO.
- Some more documentation and examples would be nice.

__init__(*subject*, *func_graph=None*, *max_iterations=30*, *track_tmps=False*, *track_consts=True*, *observation_points=None*, *init_state=None*, *init_context=None*, *state_initializer=None*, *cc=None*, *function_handler=None*, *observe_all=False*, *visited_blocks=None*, *dep_graph=True*, *observe_callback=None*, *canonical_size=8*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *interfunction_level=0*, *track_liveness=True*, *func_addr=None*, *element_limit=5*, *merge_into_tops=True*, *variable_map=None*)

**Parameters**

- **subject** (*Subject* | Block | *Block* | *Function* | str) – The subject of the analysis: a function, or a single basic block
- **func_graph** – Alternative graph for function.graph.
- **max_iterations** – The maximum number of iterations before the analysis is terminated.
- **track_tmps** – Whether or not temporary variables should be taken into consideration during the analysis.
- **observation_points** (Iterable[tuple[Literal['insn', 'node', 'stmt', 'exit'], int | tuple[int, int] | tuple[int, int, int], *ObservationPointType*]] | None) – A collection of tuples of ("node"|"insn", ins_addr, OP_TYPE) defining where reaching definitions should be copied and stored. OP_TYPE can be OP_BEFORE or OP_AFTER.
- **init_state** (*ReachingDefinitionsState* | None) – An optional initialization state. The analysis creates and works on a copy. Default to None: the analysis then initialize its own abstract state, based on the given <Subject>.
- **init_context** – If init_state is not given, this is used to initialize the context field of the initial state's CodeLocation. The only default-supported type which may go here is a tuple of integers, i.e. a callstack. Anything else requires a custom FunctionHandler.
- **cc** – Calling convention of the function.
- **function_handler** (*FunctionHandler* | None) – The function handler to update the analysis state and results on function calls.
- **observe_all** – Observe every statement, both before and after.
- **visited_blocks** – A set of previously visited blocks.
- **dep_graph** (*DepGraph* | bool | None) – An initial dependency graph to add the result of the analysis to. Set it to None to skip dependency graph generation.

- **canonical_size** – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.
- **dep_graph** – Set this to True to generate a dependency graph for the subject. It will be available as *result.dep_graph*.
- **interfunction_level** (int) – The number of functions we should recurse into. This parameter is only used if function_handler is not provided.
- **track_liveness** (bool) – Whether to track liveness information. This can consume sizeable amounts of RAM on large functions. (e.g. ~15GB for a function with 4k nodes)
- **merge_into_tops** (bool) – Merge known values into TOP if TOP is present. If True: {TOP} V {0xabc} = {TOP} If False: {TOP} V {0xabc} = {TOP, 0xabc}
- **state_initializer** (RDAStateInitializer | *None*)
- **func_addr** (*int* | *None*)
- **element_limit** (*int*)

**property observed_results:** dict[tuple[str, int, int], *LiveDefinitions*]

**property all_definitions**

**property all_uses**

**property one_result**

**property dep_graph:** *DepGraph*

**property visited_blocks**

**get_reaching_definitions_by_insn**(*ins_addr*, *op_type*)

**get_reaching_definitions_by_node**(*node_addr*, *op_type*)

**node_observe**(*node_addr*, *state*, *op_type*, *node_idx=None*)

**Parameters**

- **node_addr** (int) – Address of the node.
- **state** (*ReachingDefinitionsState*) – The analysis state.
- **op_type** (*ObservationPointType*) – Type of the observation point. Must be one of the following: OP_BEFORE, OP_AFTER.
- **node_idx** (int | None) – ID of the node. Used in AIL to differentiate blocks with the same address.

**Return type**
None

**insn_observe**(*insn_addr*, *stmt*, *block*, *state*, *op_type*)

**Parameters**

- **insn_addr** (int) – Address of the instruction.
- **stmt** (*Statement* | IRStmt) – The statement.
- **block** (*Block* | Block) – The current block.
- **state** (*ReachingDefinitionsState*) – The abstract analysis state.

- **op_type** (*ObservationPointType*) – Type of the observation point. Must be one of the following: OP_BEORE, OP_AFTER.

**Return type**
None

**stmt_observe**(*stmt_idx*, *stmt*, *block*, *state*, *op_type*)

**Parameters**

- **stmt_idx** (int)
- **stmt** (*Statement* | IRStmt)
- **block** (*Block* | Block)
- **state** (*ReachingDefinitionsState*)
- **op_type** (*ObservationPointType*)

**Return type**
None

**Returns**

**exit_observe**(*node_addr*, *exit_stmt_idx*, *block*, *state*, *node_idx=None*)

**Parameters**

- **node_addr** (*int*)
- **exit_stmt_idx** (*int*)
- **block** (Block | *Block*)
- **state** (ReachingDefinitionsState)
- **node_idx** (*int* | *None*)

**property subject**

**callsites_to**(*target*)

**Return type**
Iterable[*FunctionCallRelationships*]

**Parameters**
**target** (*int* | *str* | Function)

**class** angr.analyses.reaching_definitions.**ReachingDefinitionsModel**

Bases: object

Models the definitions, uses, and memory of a ReachingDefinitionState object

**__init__**(*func_addr=None*, *track_liveness=True*)

**Parameters**

- **func_addr** (*int* | *None*)
- **track_liveness** (*bool*)

**add_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

**kill_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

**at_new_stmt**(*codeloc*)

**Return type**
None

**Parameters**
**codeloc** (CodeLocation)

**at_new_block**(*code_loc*, *pred_codelocs*)

**Return type**
None

**Parameters**

- **code_loc** (CodeLocation)
- **pred_codelocs** (*list*[CodeLocation])

**make_liveness_snapshot**()

**Return type**
None

**find_defs_at**(*code_loc*, *op=ObservationPointType.OP_BEFORE*)

**Return type**
set[*Definition*]

**Parameters**

- **code_loc** (CodeLocation)
- **op** (*int*)

**get_defs**(*atom*, *code_loc*, *op*)

**Return type**
set[*Definition*]

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **op** (*int*)

**copy**()

**Return type**
*ReachingDefinitionsModel*

**merge**(*model*)

**Parameters**
**model** (ReachingDefinitionsModel)

**get_observation_by_insn**(*ins_addr*, *kind*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **ins_addr** (*int* | CodeLocation)
- **kind** (ObservationPointType)

**get_observation_by_node**(*node_addr*, *kind*, *node_idx=None*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **node_addr** (*int* | CodeLocation)
- **kind** (ObservationPointType)
- **node_idx** (*int* | *None*)

**get_observation_by_stmt**(*arg1*, *arg2*, *arg3=None*, ***, *block_idx=None*)

**Overloads**

- **self**, **codeloc** (CodeLocation), **kind** (ObservationPointType) → LiveDefinitions | None
- **self**, **node_addr** (int), **stmt_idx** (int), **kind** (ObservationPointType), **block_idx** (int | None)

**get_observation_by_exit**(*node_addr*, *stmt_idx*, *src_node_idx=None*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **node_addr** (*int*)
- **stmt_idx** (*int*)
- **src_node_idx** (*int* | *None*)

**class** angr.analyses.reaching_definitions.**ReachingDefinitionsState**

Bases: object

Represents the internal state of the ReachingDefinitionsAnalysis.

It contains a data class LiveDefinitions, which stores both definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

**Parameters**

- **subject** (*Subject*) – The subject being analyzed.
- **track_tmps** (bool) – Only tells whether or not temporary variables should be taken into consideration when representing the state of the analysis. Should be set to true when the analysis has counted uses and definitions for temporary variables, false otherwise.

- **analysis** (*ReachingDefinitionsAnalysis*) – The analysis that generated the state represented by this object.
- **rtoc_value** – When the targeted architecture is ppc64, the initial function needs to know the *rtoc_value*.
- **live_definitions** (*LiveDefinitions* | None)
- **canonical_size** (int) – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.
- **heap_allocator** (*HeapAllocator* | None) – Mechanism to model the management of heap memory.
- **environment** (*Environment* | None) – Representation of the environment of the analyzed program.

**Variables**

**arch** (Arch) – The architecture targeted by the program.

**__init__**(*codeloc*, *arch*, *subject*, *analysis*, *track_tmps=False*, *track_consts=False*, *rtoc_value=None*, *live_definitions=None*, *canonical_size=8*, *heap_allocator=None*, *environment=None*, *sp_adjusted=False*, *all_definitions=None*, *initializer=None*, *element_limit=5*, *merge_into_tops=True*)

**Parameters**

- **codeloc** (CodeLocation)
- **arch** (*Arch*)
- **subject** (Subject)
- **analysis** (ReachingDefinitionsAnalysis)
- **track_tmps** (*bool*)
- **track_consts** (*bool*)
- **live_definitions** (LiveDefinitions | *None*)
- **canonical_size** (*int*)
- **heap_allocator** (HeapAllocator | *None*)
- **environment** (Environment | *None*)
- **sp_adjusted** (*bool*)
- **all_definitions** (*set[Definition[A, CodeLoc]]* | *None*)
- **initializer** (RDAStateInitializer | *None*)
- **element_limit** (*int*)
- **merge_into_tops** (*bool*)

**codeloc**

**arch: Arch**

**analysis**

**all_definitions: set[*Definition*[Any, Any]]**

**heap_allocator**

**codeloc_uses:** set[*Definition*[Any, Any]]

**exit_observed:** bool

**live_definitions**

**top**(*bits*)

**Parameters**
**bits** (*int*)

**is_top**(**args*)

**heap_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int* | HeapAddress)

**static is_heap_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**static get_heap_offset**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**stack_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int*)

**is_stack_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**get_stack_offset**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**annotate_with_def**(*symvar*, *definition*)

**Parameters**

- **symvar** (TypeVar(MVType, bound= BV | FP))
- **definition** (*Definition*[Any, Any])

**Return type**
TypeVar(MVType, bound= BV | FP)

**Returns**

**annotate_mv_with_def**(*mv*, *definition*)

**Return type**
*MultiValues*[TypeVar(MVType, bound= BV | FP)]

**Parameters**

- **mv** (MultiValues*[MVType]*)
- **definition** (Definition*[A, CodeLoc]*)

**extract_defs**(*symvar*)

**Return type**
Iterator[*Definition*[Any, Any]]

**Parameters**
**symvar** (*Base*)

**property tmps**

**property tmp_uses**

**property registers: *MultiValuedMemory***

**property register_uses**

**property stack: *MultiValuedMemory***

**property stack_uses**

**property heap: *MultiValuedMemory***

**property heap_uses**

**property memory: *MultiValuedMemory***

**property memory_uses**

**property others: dict[*Atom*, *MultiValues*]**

**property uses_by_codeloc**

**get_sp**()

**Return type**
int

**get_stack_address**(*offset*)

**Return type**
int | None

**Parameters**
**offset** (*Base*)

**property environment**

**property dep_graph**

**copy**(*discard_tmpdefs=False*)

**Return type**
Self

**merge**(**others*)

**Return type**
tuple[Self, bool]

**Parameters**
**others** (*Self*)

**compare**(*other*)

**Return type**
bool

**Parameters**
**other** (ReachingDefinitionsState)

**move_codelocs**(*new_codeloc*)

**Return type**
None

**Parameters**
**new_codeloc** (CodeLocation)

**kill_definitions**(*atom*)

Overwrite existing definitions w.r.t 'atom' with a dummy definition instance. A dummy definition will not be removed during simplification.

**Return type**
None

**Parameters**
**atom** (Atom)

**kill_and_add_definition**(*atom*, *data*, *dummy=False*, *tags=None*, *endness=None*, *annotated=False*, *uses=None*, *override_codeloc=None*)

**Return type**
tuple[*MultiValues* | None, set[*Definition*[TypeVar(A, bound= *Atom*), TypeVar(CodeLoc, bound= CodeLocation | AILCodeLocation)]]]

**Parameters**

- **atom** (Atom)
- **data** (MultiValues)

- **tags** (*set[Tag] | None*)
- **annotated** (*bool*)
- **uses** (*set[Definition[A, CodeLoc]] | None*)
- **override_codeloc** (*CodeLocation | None*)

**add_use**(*atom*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (*Atom*)
- **expr** (*Any | None*)

**add_use_by_def**(*definition*, *expr=None*)

**Return type**
None

**Parameters**

- **definition** (*Definition[A, CodeLoc]*)
- **expr** (*Any | None*)

**add_tmp_use**(*tmp*, *expr=None*)

**Return type**
None

**Parameters**

- **tmp** (*int*)
- **expr** (*Any | None*)

**add_tmp_use_by_defs**(*defs*, *expr=None*)

**Return type**
None

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**add_register_use**(*reg_offset*, *size*, *expr=None*)

**Return type**
None

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)
- **expr** (*Any | None*)

**add_register_use_by_defs**(*defs*, *expr=None*)

**Return type**
None

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**add_stack_use**(*stack_offset*, *size*, *expr=None*)

**Return type**
None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **expr** (*Any | None*)

**add_stack_use_by_defs**(*defs*, *expr=None*)

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**add_heap_use**(*heap_offset*, *size*, *expr=None*)

**Return type**
None

**Parameters**

- **heap_offset** (*int*)
- **size** (*int*)
- **expr** (*Any | None*)

**add_heap_use_by_defs**(*defs*, *expr=None*)

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**add_memory_use_by_def**(*definition*, *expr=None*)

**Parameters**

- **definition** (*Definition[A, CodeLoc]*)
- **expr** (*Any | None*)

**add_memory_use_by_defs**(*defs*, *expr=None*)

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**get_definitions**(*atom*)

**Return type**
set[*Definition*[*Atom*, TypeVar(CodeLoc, bound= CodeLocation | AILCodeLocation)]]

**Parameters**
**atom** (*Atom* | *Definition*[*Atom*, *CodeLoc*] | *Iterable*[*Atom*] | *Iterable*[*Definition*[*Atom*, *CodeLoc*]] | MultiValues)

**get_values**(*spec*)

**Return type**
*MultiValues* | None

**Parameters**
**spec** (*A* | *Definition*[*A*, *CodeLoc*] | *Iterable*[*A*])

**get_one_value**(*spec*, *strip_annotations=False*)

**Return type**
BV | None

**Parameters**

- **spec** (*A* | *Definition*[*A*, *CodeLoc*] | *Iterable*[*A*] | *Iterable*[*Definition*[*A*, *CodeLoc*]])
- **strip_annotations** (*bool*)

**get_concrete_value**(*spec*, *cast_to=<class 'int'>*)

**Overloads**

- **self**, **spec** (Atom | Definition[Atom, CodeLoc] | Iterable[Atom]), **cast_to** (type[int]) → int | None
- **self**, **spec** (Atom | Definition[Atom, CodeLoc] | Iterable[Atom]), **cast_to** (type[bytes]) → bytes | None

**Parameters**

- **spec** (*Atom* | *Definition*[*Atom*, *CodeLoc*] | *Iterable*[*Atom*])
- **cast_to** (type[*int* | bytes])

**Return type**
int | *bytes* | None

**mark_guard**(*target*)

**mark_const**(*value*, *size*)

**Parameters**

- **value** (*int*)
- **size** (*int*)

**downsize**()

**deref**(*pointer*, *size*, *endness=Endness.BE*)

**Overloads**

- **self**, **pointer** (int | claripy.ast.BV | HeapAddress | SpOffset), **size** (int | DerefSize), **endness** (archinfo.Endness) → MemoryLocation | None

- **self**, **pointer** (MultiValues | A | Definition | Iterable[A] | Iterable[Definition[A, CodeLoc]]), **size** (int | DerefSize), **endness** (archinfo.Endness) → set[MemoryLocation]

**Parameters**

- **pointer** (*MultiValues[BV] | Atom | Definition[Atom, CodeLoc] | Iterable[Atom] | Iterable[Definition[Atom, CodeLoc]] | int | BV | HeapAddress | SpOffset*)
- **size** (*int | DerefSize*)
- **endness** (*Endness*)

**class** angr.analyses.reaching_definitions.**Register**

Bases: *Atom*

Represents a given CPU register.

As an IR abstracts the CPU design to target different architectures, registers are represented as a separated memory space. Thus a register is defined by its offset from the base of this memory and its size.

**Variables**

- **reg_offset** (*int*) – The offset from the base to define its place in the memory bloc.
- **size** (*int*) – The size, in number of bytes.

**reg_offset**

**arch**

**property name:** str

**class** angr.analyses.reaching_definitions.**Tmp**

Bases: *Atom*

Represents a variable used by the IR to store intermediate values.

**tmp_idx**

angr.analyses.reaching_definitions.**get_all_definitions**(*region*)

**Return type**
set[*Definition*]

**Parameters**
**region** (MultiValuedMemory)

## Submodules

| |
|---|
| *call_trace* |
| *dep_graph* |
| *engine_ail* |
| *engine_vex* |
| *external_codeloc* |
| *function_handler* |
| *function_handler_library* |
| *heap_allocator* |
| *rd_initializer* |
| *rd_state* |

continues on next page

Table 38 – continued from previous page

| |
|---|
| *reaching_definitions* |
| *subject* |

### angr.analyses.reaching_definitions.call_trace

**class** angr.analyses.reaching_definitions.call_trace.**CallSite**

Bases: object

Describes a call site on a CFG.

**__init__**(*caller_func_addr*, *block_addr*, *callee_func_addr*)

**Parameters**

- **caller_func_addr** (*int*)
- **block_addr** (*int | None*)
- **callee_func_addr** (*int*)

**caller_func_addr**

**callee_func_addr**

**block_addr**

**class** angr.analyses.reaching_definitions.call_trace.**CallTrace**

Bases: object

Describes a series of functions calls to get from one function (current_function_address()) to another function or a basic block (self.target).

**__init__**(*target*)

**Parameters**
**target** (*int*)

**target**

**callsites: list[*CallSite*]**

**current_function_address**()

**Return type**
int

**step_back**(*caller_func_addr*, *block_addr*, *callee_func_addr*)

**Return type**
*CallTrace*

**Parameters**

- **caller_func_addr** (*int*)
- **block_addr** (*int | None*)

**includes_function**(*func_addr*)

**Return type**
bool

**Parameters**
**func_addr** (*int*)

`copy()`

**Return type**
*CallTrace*

### angr.analyses.reaching_definitions.dep_graph

`class angr.analyses.reaching_definitions.dep_graph.DepGraph`

Bases: object

The representation of a dependency graph: a directed graph, where nodes are definitions, and edges represent uses.

Mostly a wrapper around a <networkx.DiGraph>.

`__init__(graph=None)`

**Parameters**
**graph** – A graph where nodes are definitions, and edges represent uses.

`property graph: networkx.DiGraph[Definition]`

`add_node(node)`

**Parameters**
**node** (*Definition*) – The definition to add to the definition-use graph.

**Return type**
None

`add_edge(source, destination, **labels)`

The edge to add to the definition-use graph. Will create nodes that are not yet present.

**Parameters**

- **source** (*Definition*) – The "source" definition, used by the "destination".
- **destination** (*Definition*) – The "destination" definition, using the variable defined by "source".
- **labels** – Optional keyword arguments to represent edge labels.

**Return type**
None

`nodes()`

**Return type**
Iterator[*Definition*]

`predecessors(node)`

**Parameters**
**node** (*Definition*) – The definition to get the predecessors of.

**Return type**
Iterator[*Definition*]

`transitive_closure`(*definition*)

Compute the "transitive closure" of a given definition. Obtained by transitively aggregating the ancestors of this definition in the graph.

Note: Each definition is memoized to avoid any kind of recomputation across the lifetime of this object.

**Parameters**
**definition** – The Definition to get transitive closure for.

**Returns**
A graph of the transitive closure of the given definition.

**Return type**
networkx.DiGraph[*Definition*[*Atom*]]

`contains_atom`(*atom*)

**Return type**
bool

**Parameters**
**atom** (Atom)

`add_dependencies_for_concrete_pointers_of`(*values*, *definition*, *cfg*, *loader*)

When a given definition holds concrete pointers, make sure the <MemoryLocation>s they point to are present in the dependency graph; Adds them if necessary.

**Parameters**

- **values** (Iterable[Base | int])
- **definition** (*Definition*) – The definition which has data that can contain concrete pointers.
- **cfg** (*CFGModel* | None) – The CFG, containing information about memory data.
- **loader** (Loader)

`find_definitions`(***kwargs*)

Filter the definitions present in the graph based on various criteria. Parameters can be any valid keyword args to *DefinitionMatchPredicate*

**Overloads**

- **self**, **kind** (type[A]), **kwargs** (Any) → list[Definition[A]]
- **self**, **kind** (Literal[AtomKind.REGISTER]), **kwargs** (Any) → list[Definition[Register]]
- **self**, **kind** (Literal[AtomKind.MEMORY]), **kwargs** (Any) → list[Definition[MemoryLocation]]
- **self**, **kind** (Literal[AtomKind.TMP]), **kwargs** (Any) → list[Definition[Tmp]]
- **self**, **kind** (Literal[AtomKind.CONSTANT]), **kwargs** (Any) → list[Definition[ConstantSrc]]
- **self**, **kind** (Literal[AtomKind.GUARD]), **kwargs** (Any) → list[Definition[GuardUse]]
- **self**, **reg_name** (int | str), **kwargs** (Any) → list[Definition[Register]]
- **self**, **stack_offset** (int), **kwargs** (Any) → list[Definition[MemoryLocation]]
- **self**, **const_val** (int), **kwargs** (Any) → list[Definition[ConstantSrc]]

**Return type**
list[Definition]

`find_all_predecessors`(*starts*, ***kwargs*)

Filter the ancestors of the given start node or nodes that match various criteria. Parameters can be any valid keyword args to *DefinitionMatchPredicate*

**Overloads**

- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **kind** (type[A]), **kwargs** (Any) → list[Definition[A]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **kind** (Literal[AtomKind.REGISTER]), **kwargs** (Any) → list[Definition[Register]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **kind** (Literal[AtomKind.MEMORY]), **kwargs** (Any) → list[Definition[MemoryLocation]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **kind** (Literal[AtomKind.TMP]), **kwargs** (Any) → list[Definition[Tmp]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **kind** (Literal[AtomKind.CONSTANT]), **kwargs** (Any) → list[Definition[ConstantSrc]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **kind** (Literal[AtomKind.GUARD]), **kwargs** (Any) → list[Definition[GuardUse]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **reg_name** (int | str), **kwargs** (Any) → list[Definition[Register]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **stack_offset** (int), **kwargs** (Any) → list[Definition[MemoryLocation]]
- **self**, **starts** (Definition[Atom] | Iterable[Definition[Atom]]), **const_val** (int), **kwargs** (Any) → list[Definition[ConstantSrc]]

`find_all_successors`(*starts*, ***kwargs*)

Filter the descendents of the given start node or nodes that match various criteria. Parameters can be any valid keyword args to *DefinitionMatchPredicate*

**Return type**
list[*Definition*]

**Parameters**
**starts** (Definition | *Iterable*[Definition])

`find_path`(*starts*, *ends*, ***kwargs*)

Find a path between the given start node or nodes and the given end node or nodes. All the intermediate steps in the path must match the criteria given in kwargs. The kwargs can be any valid parameters to *DefinitionMatchPredicate*.

This algorithm has exponential time and space complexity. Use at your own risk. Want to do better? Do it yourself or use networkx and eat the cost of indirection and/or cloning.

**Return type**
tuple[*Definition*, ...] | None

**Parameters**

- **starts** (Definition | *Iterable*[Definition])
- **ends** (Definition | *Iterable*[Definition])

**find_paths**(*starts*, *ends*, ***kwargs*)

Find all non-overlapping simple paths between the given start node or nodes and the given end node or nodes. All the intermediate steps in the path must match the criteria given in kwargs. The kwargs can be any valid parameters to *DefinitionMatchPredicate*.

This algorithm has exponential time and space complexity. Use at your own risk. Want to do better? Do it yourself or use networkx and eat the cost of indirection and/or cloning.

**Return type**
Iterator[tuple[*Definition*, ...]]

**Parameters**

- **starts** (Definition | *Iterable*[Definition])
- **ends** (Definition | *Iterable*[Definition])

### angr.analyses.reaching_definitions.engine_ail

**class** angr.analyses.reaching_definitions.engine_ail.**SimEngineRDAIL**

Bases: *SimEngineNostmtAIL*[*ReachingDefinitionsState*, *MultiValues*[BV | FP], None, *ReachingDefinitionsState*]

**__init__**(*project*, *function_handler*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *bp_as_gpr=False*, *variable_map=None*)

**Parameters**

- **function_handler** (FunctionHandler)
- **bp_as_gpr** (*bool*)

**process**(*state*, ***, *dep_graph=None*, *visited_blocks=None*, *block=None*, *fail_fast=False*, *whitelist=None*, ***kwargs*)

### angr.analyses.reaching_definitions.engine_vex

**class** angr.analyses.reaching_definitions.engine_vex.**SimEngineRDVEX**

Bases: *SimEngineNostmtVEX*[*ReachingDefinitionsState*, *MultiValues*[BV | FP], *ReachingDefinitionsState*]

Implements the VEX execution engine for reaching definition analysis.

**__init__**(*project*, *function_handler*, *functions*)

**Parameters**

- **function_handler** (FunctionHandler)
- **functions** (FunctionManager)

**process**(*state*, ***, *block=None*, *fail_fast=False*, *visited_blocks=None*, *dep_graph=None*, *whitelist=None*, ***kwargs*)

### angr.analyses.reaching_definitions.external_codeloc

### angr.analyses.reaching_definitions.function_handler

angr.analyses.reaching_definitions.function_handler.**get_exit_livedefinitions**(*func*, *rda_model*)

Get LiveDefinitions at all exits of a function, merge them, and return.

**Parameters**

- **func** (Function)
- **rda_model** (ReachingDefinitionsModel)

**class** angr.analyses.reaching_definitions.function_handler.**FunctionEffect**

Bases: object

A single effect that a function summary may apply to the state. This is largely an implementation detail; use *FunctionCallData.depends* instead.

**dest:** *Atom* | None

**sources:** set[*Atom*]

**value:** *MultiValues* | None = None

**sources_defns:** set[*Definition*] | None = None

**apply_at_callsite:** bool = False

**tags:** set[*Tag*] | None = None

**__init__**(*dest*, *sources*, *value=None*, *sources_defns=None*, *apply_at_callsite=False*, *tags=None*)

**Parameters**

- **dest** (*Atom | None*)
- **sources** (*set[Atom]*)
- **value** (*MultiValues | None*)
- **sources_defns** (*set[Definition] | None*)
- **apply_at_callsite** (*bool*)
- **tags** (*set[Tag] | None*)

**Return type**
None

**class** angr.analyses.reaching_definitions.function_handler.**FunctionCallData**

Bases: object

A bundle of intermediate data used when computing the sum effect of a function during ReachingDefinitionsAnalysis.

RDA engine contract:

- Construct one of these before calling *FunctionHandler.handle_function*. Fill it with as many fields as you can realistically provide without duplicating effort.
- Provide *callsite_codeloc* as either the call statement (AIL) or the default exit of the default statement of the calling block (VEX)
- Provide *function_codeloc* as the callee address with *stmt_idx=0* `.

Function handler contract:

- If redefine_locals is unset, do not adjust any artifacts of the function call abstraction, such as the stack pointer, the caller saved registers, etc.
- If caller_will_handle_single_ret is set, and there is a single entry in *ret_atoms*, do not apply to the state effects modifying this atom. Instead, set *ret_values* and *ret_values_deps* to the values and deps which are used constructing these values.

**callsite_codeloc**: *CodeLocation*

**function_codeloc**: *CodeLocation*

**address_multi**: *MultiValues*[BV | FP] | None

**address**: int | None = None

**symbol**: Symbol | None = None

**function**: *Function* | None = None

**name**: str | None = None

**cc**: *SimCC* | None = None

**prototype**: *SimTypeFunction* | None = None

**args_atoms**: list[set[*Atom*]] | None = None

**args_values**: list[*MultiValues*[BV | FP]] | None = None

**ret_atoms**: set[*Atom*] | None = None

**redefine_locals**: bool = True

**visited_blocks**: set[int] | None = None

**effects**: list[*FunctionEffect*]

**ret_values**: *MultiValues*[BV | FP] | None = None

**ret_values_deps**: set[*Definition*] | None = None

**caller_will_handle_single_ret**: bool = False

**guessed_cc**: bool = False

**guessed_prototype**: bool = False

**retaddr_popped**: bool = False

**has_clobbered**(*dest*)

Determines whether the given atom already has effects applied

**Return type**
bool

**Parameters**
**dest** (Atom)

**depends**(*dest*, **sources*, *value=None*, *apply_at_callsite=False*, *tags=None*)

Mark a single effect of the current function, including the atom being modified, the input atoms on which that output atom depends, the precise (or imprecise!) value to store, and whether the effect should be applied during the function or afterwards, at the callsite.

The tags are used to annotate the Definition of the Atom that will be created, when the function effects are applied to the state.

The atom being modified may be None to mark uses of the source atoms which do not have any explicit sinks.

**Parameters**

- **dest** (Atom | *Iterable*[Atom] | *None*)
- **sources** (Atom | *Iterable*[Atom])
- **value** (MultiValues | *BV* | bytes | *int* | *None*)
- **apply_at_callsite** (*bool*)
- **tags** (*set*[Tag] | *None*)

**reset_prototype**(*prototype*, *state*, *soft_reset=False*)

**Return type**
set[*Atom*]

**Parameters**

- **prototype** (SimTypeFunction)
- **state** (ReachingDefinitionsState)
- **soft_reset** (*bool*)

**__init__**(*callsite_codeloc*, *function_codeloc*, *address_multi*, *address=None*, *symbol=None*, *function=None*, *name=None*, *cc=None*, *prototype=None*, *args_atoms=None*, *args_values=None*, *ret_atoms=None*, *redefine_locals=True*, *visited_blocks=None*, *effects=<factory>*, *ret_values=None*, *ret_values_deps=None*, *caller_will_handle_single_ret=False*, *guessed_cc=False*, *guessed_prototype=False*, *retaddr_popped=False*)

**Parameters**

- **callsite_codeloc** (CodeLocation)
- **function_codeloc** (CodeLocation)
- **address_multi** (MultiValues[*BV* | *FP*] | *None*)
- **address** (*int* | *None*)
- **symbol** (*Symbol* | *None*)
- **function** (Function | *None*)
- **name** (*str* | *None*)
- **cc** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **args_atoms** (*list*[*set*[Atom]] | *None*)
- **args_values** (*list*[MultiValues[*BV* | *FP*]] | *None*)
- **ret_atoms** (*set*[Atom] | *None*)

- **redefine_locals** (*bool*)
- **visited_blocks** (*set[int] | None*)
- **effects** (*list[FunctionEffect]*)
- **ret_values** (*MultiValues[BV | FP] | None*)
- **ret_values_deps** (*set[Definition] | None*)
- **caller_will_handle_single_ret** (*bool*)
- **guessed_cc** (*bool*)
- **guessed_prototype** (*bool*)
- **retaddr_popped** (*bool*)

**Return type**
None

**class** angr.analyses.reaching_definitions.function_handler.**FunctionCallDataUnwrapped**

Bases: *FunctionCallData*

A subclass of FunctionCallData which asserts that many of its members are non-None at construction time. Typechecks be gone!

**address_multi:** *MultiValues*

**__init__**(*inner*)

**Parameters**
**inner** (FunctionCallData)

**static decorate**(*wrapper*, *, *wrapped=<function FunctionCallDataUnwrapped.decorate>*, *assigned=('__module__', '__name__', '__qualname__', '__doc__', '__annotations__', '__type_params__')*, *updated=('__dict__', )*)

Update a wrapper function to look like the wrapped function

wrapper is the function to be updated wrapped is the original function assigned is a tuple naming the attributes assigned directly from the wrapped function to the wrapper function (defaults to functools.WRAPPER_ASSIGNMENTS) updated is a tuple naming the attributes of the wrapper that are updated with the corresponding attribute from the wrapped function (defaults to functools.WRAPPER_UPDATES)

**class** angr.analyses.reaching_definitions.function_handler.**FunctionCallRelationships**

Bases: object

Produced by the function handler, provides associated callsite info and function input/output definitions.

**callsite:** *CodeLocation*

**target:** int | None

**args_defns:** list[set[*Definition*]]

**other_input_defns:** set[*Definition*]

**ret_defns:** set[*Definition*]

**other_output_defns:** set[*Definition*]

**__init__**(*callsite*, *target*, *args_defns*, *other_input_defns*, *ret_defns*, *other_output_defns*)

**Parameters**

- **callsite** (CodeLocation)
- **target** (*int | None*)
- **args_defns** (*list[set[Definition]]*)
- **other_input_defns** (*set[Definition]*)
- **ret_defns** (*set[Definition]*)
- **other_output_defns** (*set[Definition]*)

**Return type**
None

**class** angr.analyses.reaching_definitions.function_handler.**FunctionHandler**

Bases: object

A mechanism for summarizing a function call's effect on a program for ReachingDefinitionsAnalysis.

**__init__**(*interfunction_level=0*, *extra_impls=None*)

**Parameters**

- **interfunction_level** (int) – Maximum depth in to continue local function exploration
- **extra_impls** (Iterable[type[*FunctionHandler*]] | None) – FunctionHandler classes to implement beyond what's implemented in function_handler_library

**interfunction_level:** int

**hook**(*analysis*)

Attach this instance of the function handler to an instance of RDA.

**Return type**
*FunctionHandler*

**Parameters**
**analysis** (ReachingDefinitionsAnalysis)

**make_function_codeloc**(*target*, *callsite*, *callsite_func_addr*)

The RDA engine will call this function to transform a callsite CodeLocation into a callee CodeLocation.

**Parameters**

- **target** (*int | MultiValues | None*)
- **callsite** (CodeLocation)
- **callsite_func_addr** (*int | None*)

**handle_function**(*state*, *data*)

The main entry point for the function handler. Called with a RDA state and a FunctionCallData, it is expected to update the state and the data as per the contracts described on FunctionCallData.

You can override this method to take full control over how data is processed, or override any of the following to use the higher-level interface (data.depends()):

- *handle_impl_<function name>* - used for *<function name>*.
- *handle_local_function* - used for any function (excluding plt stubs) whose address is inside the main binary.

- *handle_external_function* - used for any function or plt stub whose address is outside the main binary.
- *handle_indirect_function* - used for any function whose target cannot be resolved.
- *handle_generic_function* - used as a default if none of the above are overridden.

Each of them take the same signature as *handle_function*.

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_generic_function**(*state*, *data*)

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_indirect_function**(*state*, *data*)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_local_function**(*state*, *data*)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**handle_external_function**(*state*, *data*)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**recurse_analysis**(*state*, *data*)

Precondition: `data.function` MUST NOT BE NONE in order to call this method.

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallData)

**static c_args_as_atoms**(*state*, *cc*, *prototype*)

**Return type**
list[set[*Atom*]]

**Parameters**

- **state** (ReachingDefinitionsState)
- **cc** (SimCC)
- **prototype** (SimTypeFunction)

**static c_return_as_atoms**(*state*, *cc*, *prototype*)

**Return type**
set[*Atom*]

**Parameters**

- **state** (ReachingDefinitionsState)
- **cc** (SimCC)
- **prototype** (SimTypeFunction)

**static caller_saved_regs_as_atoms**(*state*, *cc*)

**Return type**
set[*Register*]

**Parameters**

- **state** (ReachingDefinitionsState)
- **cc** (SimCC)

**static stack_pointer_as_atom**(*state*)

**Return type**
*Register*

### angr.analyses.reaching_definitions.function_handler_library

**class** angr.analyses.reaching_definitions.function_handler_library.**EnvironAtom**

Bases: *Atom*

**size**

**class** angr.analyses.reaching_definitions.function_handler_library.**ExecveAtom**

Bases: *Atom*

**size**

**class** angr.analyses.reaching_definitions.function_handler_library.**LibcHandlers**

Bases: *LibcStdlibHandlers*, *LibcStdioHandlers*, *LibcUnistdHandlers*, *LibcStringHandlers*

**class** angr.analyses.reaching_definitions.function_handler_library.**StdinAtom**

Bases: *Atom*

**size**

**class** angr.analyses.reaching_definitions.function_handler_library.**StdoutAtom**

Bases: *Atom*

**size**

class angr.analyses.reaching_definitions.function_handler_library.**SystemAtom**

Bases: *Atom*

**size**

### Submodules


*stdio*
*stdlib*
*string*
*unistd*


### angr.analyses.reaching_definitions.function_handler_library.stdio

class angr.analyses.reaching_definitions.function_handler_library.stdio.**StdoutAtom**

Bases: *Atom*

**size**

class angr.analyses.reaching_definitions.function_handler_library.stdio.**StdinAtom**

Bases: *Atom*

**size**

angr.analyses.reaching_definitions.function_handler_library.stdio.**parse_format_string**(*format_string*)

**Return type**
tuple[list[str | int], list[*SimType*], list[str]]

**Parameters**
**format_string** (*str*)

class angr.analyses.reaching_definitions.function_handler_library.stdio.**LibcStdioHandlers**

Bases: *FunctionHandler*

**handle_impl_printf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_dprintf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_fprintf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_sprintf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_snprintf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl___sprintf_chk**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl___snprintf_chk**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_scanf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl___isoc99_scanf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_sscanf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl___isoc99_sscanf**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_fgets**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_fgetc**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_getchar**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_getc**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_fread**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_fwrite**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

angr.analyses.reaching_definitions.function_handler_library.stdio.**handle_printf**(*state*, *data*, *fmt_idx*)

**Return type**
tuple[*MultiValues* | None, Iterable[*Atom*]]

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallDataUnwrapped)
- **fmt_idx** (*int*)

angr.analyses.reaching_definitions.function_handler_library.stdio.**handle_scanf**(*state*, *data*, *fmt_idx*, *source_atoms*)

**Parameters**

- **state** (ReachingDefinitionsState)
- **data** (FunctionCallDataUnwrapped)
- **fmt_idx** (*int*)
- **source_atoms** (*Iterable[Atom]*)

### angr.analyses.reaching_definitions.function_handler_library.stdlib

**class** angr.analyses.reaching_definitions.function_handler_library.stdlib.**EnvironAtom**

Bases: *Atom*

**size**

**class** angr.analyses.reaching_definitions.function_handler_library.stdlib.**SystemAtom**

Bases: *Atom*

**size**

**class** angr.analyses.reaching_definitions.function_handler_library.stdlib.**ExecveAtom**

Bases: *Atom*

**size**

**class** angr.analyses.reaching_definitions.function_handler_library.stdlib.**LibcStdlibHandlers**

Bases: *FunctionHandler*

**handle_impl_atoi**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_malloc**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_calloc**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_getenv**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_setenv**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_system**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_popen**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_execl**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_execve**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

### angr.analyses.reaching_definitions.function_handler_library.string

**class** angr.analyses.reaching_definitions.function_handler_library.string.**LibcStringHandlers**

Bases: *FunctionHandler*

**handle_impl_strcat**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_strncat**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_strlen**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_strcpy**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_strncpy**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_strdup**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

handle_impl_memcpy(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[*FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped*], None]*)

handle_impl_memset(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[*FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped*], None]*)

handle_impl_strtok(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[*FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped*], None]*)

handle_impl_strtok_r(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[*FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped*], None]*)

handle_impl_strstr(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[*FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped*], None]*)

**handle_impl_strcasestr**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_strchr**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_imple_strrchr**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_strchrnul**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

### angr.analyses.reaching_definitions.function_handler_library.unistd

**class** angr.analyses.reaching_definitions.function_handler_library.unistd.**FDAtom**

Bases: *Atom*

**size**

**class** angr.analyses.reaching_definitions.function_handler_library.unistd.**LibcUnistdHandlers**

Bases: *FunctionHandler*

**handle_impl_read**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_recv**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_recvfrom**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_write**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

**handle_impl_send**(*f*)

Decorate a function handler method with this to make it take a FunctionCallDataUnwrapped instead of a FunctionCallData.

**Return type**
Callable[[*FunctionHandler*, *ReachingDefinitionsState*, *FunctionCallData*], None]

**Parameters**
**f** (*Callable[[FunctionHandler, ReachingDefinitionsState, FunctionCallDataUnwrapped], None]*)

### angr.analyses.reaching_definitions.heap_allocator

**class** angr.analyses.reaching_definitions.heap_allocator.**HeapAllocator**

Bases: object

A simple modelisation to help represent heap memory management during a <ReachingDefinitionsAnalysis>: - Act as if allocations were always done in consecutive memory segments; - Take care of the size not to screw potential pointer arithmetic (avoid overlapping segments).

The content of the heap itself is modeled using a <KeyedRegion> attribute in the <LiveDefinitions> state; This class serves to generate consistent heap addresses to be used by the aforementioned.

*Note:* This has **NOT** been made to help detect heap vulnerabilities.

**__init__**(*canonical_size*)

**Parameters**
**canonical_size** (int) – The concrete size an <UNKNOWN_SIZE> defaults to.

**allocate**(*size*)

Gives an address for a new memory chunk of <size> bytes.

**Parameters**
**size** (int | *UnknownSize*) – The requested size for the chunk, in number of bytes.

**Return type**
*HeapAddress*

**Returns**
The address of the chunk.

**free**(*address*)

Mark the chunk pointed by <address> as freed.

**Parameters**
**address** (*Undefined* | *HeapAddress*) – The address of the chunk to free.

**property allocated_addresses**

The list of addresses that are currently allocated on the heap.

**Type**
return

### angr.analyses.reaching_definitions.rd_initializer

**class** angr.analyses.reaching_definitions.rd_initializer.**RDAStateInitializer**

Bases: object

This class acts as the basic implementation for the logic that initializes the base state for the reaching definitions analysis.

It also defines the _interface_ that a state initializer should implement, if the language/runtime being analyzed requires more complicated logic to set up the state.

This code/logic was previously part of the ReachingDefinitionsState class, but this was moved here to separate these two concerns, and allow easily changing the initialization logic without having to change the state class.

**__init__**(*arch*, *project=None*)

**Parameters**
**arch** (*Arch*)

**initialize_function_state**(*state*, *cc*, *func_addr*, *rtoc_value=None*)

This is the entry point to the state initialization logic. It will be called during the initialization of an ReachingDefinitionsState, if the state was freshly created (without existing live_definitions)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **cc** (SimCC | *None*)
- **func_addr** (*int*)
- **rtoc_value** (*int* | *None*)

**initialize_all_function_arguments**(*state*, *func_addr*, *ex_loc*, *cc*, *prototype*)

This method handles the setup for _all_ arguments of a function.

The default implementation uses the calling convention to extract the argument locations, associates them with the type, and passes them to the logic for one argument.

You probably don't need to override this

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **func_addr** (*int*)
- **ex_loc** (ExternalCodeLocation)
- **cc** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)

**initialize_one_function_argument**(*state*, *func_addr*, *ex_loc*, *argument_location*, *argument_type=None*)

This method handles the setup for _one_ argument of a function. This is the main method to override for custom initialization logic.

The default implementation initializes only the argument location itself, but the signature allows this to support extra logic based on the _type_ of the argument as well.

For example if the argument is a pointer to something, the default implementation would only set up the register with the value TOP.

A custom implementation could instead dedicate some memory somewhere (e.g. on the heap), setup whatever object is being pointed to, and then put the actual pointer to this inside the register

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **func_addr** (*int*)
- **ex_loc** (ExternalCodeLocation)
- **argument_location** (SimFunctionArgument)

- **argument_type** (SimType | *None*)

**initialize_stack_pointer**(*state*, *_func_addr*, *ex_loc*)

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **_func_addr** (*int*)
- **ex_loc** (ExternalCodeLocation)

**initialize_architectural_state**(*state*, *func_addr*, *ex_loc*, *rtoc_value=None*)

Some architectures require initialization that is specific to that architecture.

Override this if you need to add support for an architecture that requires this, and isn't covered yet

**Return type**
None

**Parameters**

- **state** (ReachingDefinitionsState)
- **func_addr** (*int*)
- **ex_loc** (ExternalCodeLocation)
- **rtoc_value** (*int* | *None*)

### angr.analyses.reaching_definitions.rd_state

**class** angr.analyses.reaching_definitions.rd_state.**ReachingDefinitionsState**

Bases: object

Represents the internal state of the ReachingDefinitionsAnalysis.

It contains a data class LiveDefinitions, which stores both definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

**Parameters**

- **subject** (*Subject*) – The subject being analyzed.
- **track_tmps** (bool) – Only tells whether or not temporary variables should be taken into consideration when representing the state of the analysis. Should be set to true when the analysis has counted uses and definitions for temporary variables, false otherwise.
- **analysis** (*ReachingDefinitionsAnalysis*) – The analysis that generated the state represented by this object.
- **rtoc_value** – When the targeted architecture is ppc64, the initial function needs to know the *rtoc_value*.
- **live_definitions** (*LiveDefinitions* | None)
- **canonical_size** (int) – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.
- **heap_allocator** (*HeapAllocator* | None) – Mechanism to model the management of heap memory.

- **environment** (*Environment* | None) – Representation of the environment of the analyzed program.

**Variables**
**arch** (Arch) – The architecture targeted by the program.

**__init__**(*codeloc*, *arch*, *subject*, *analysis*, *track_tmps=False*, *track_consts=False*, *rtoc_value=None*, *live_definitions=None*, *canonical_size=8*, *heap_allocator=None*, *environment=None*, *sp_adjusted=False*, *all_definitions=None*, *initializer=None*, *element_limit=5*, *merge_into_tops=True*)

**Parameters**

- **codeloc** (CodeLocation)
- **arch** (*Arch*)
- **subject** (Subject)
- **analysis** (ReachingDefinitionsAnalysis)
- **track_tmps** (*bool*)
- **track_consts** (*bool*)
- **live_definitions** (LiveDefinitions | *None*)
- **canonical_size** (*int*)
- **heap_allocator** (HeapAllocator | *None*)
- **environment** (Environment | *None*)
- **sp_adjusted** (*bool*)
- **all_definitions** (*set[Definition[A, CodeLoc]]* | *None*)
- **initializer** (RDAStateInitializer | *None*)
- **element_limit** (*int*)
- **merge_into_tops** (*bool*)

**codeloc**

**arch**: Arch

**analysis**

**all_definitions**: set[*Definition*[Any, Any]]

**heap_allocator**

**codeloc_uses**: set[*Definition*[Any, Any]]

**exit_observed**: bool

**live_definitions**

**top**(*bits*)

**Parameters**
**bits** (*int*)

**is_top**(**args*)

**heap_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int | HeapAddress*)

**static is_heap_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**static get_heap_offset**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**stack_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int*)

**is_stack_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**get_stack_offset**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**annotate_with_def**(*symvar*, *definition*)

**Parameters**

- **symvar** (TypeVar(MVType, bound= BV | FP))
- **definition** (*Definition*[Any, Any])

**Return type**
TypeVar(MVType, bound= BV | FP)

**Returns**

**annotate_mv_with_def**(*mv*, *definition*)

**Return type**
*MultiValues*[TypeVar(MVType, bound= BV | FP)]

**Parameters**

- **mv** (*MultiValues[MVType]*)
- **definition** (*Definition[A, CodeLoc]*)

**extract_defs**(*symvar*)

**Return type**
Iterator[*Definition*[Any, Any]]

**Parameters**
**symvar** (*Base*)

**property tmps**

**property tmp_uses**

**property registers: *MultiValuedMemory***

**property register_uses**

**property stack: *MultiValuedMemory***

**property stack_uses**

**property heap: *MultiValuedMemory***

**property heap_uses**

**property memory: *MultiValuedMemory***

**property memory_uses**

**property others: dict[*Atom*, *MultiValues*]**

**property uses_by_codeloc**

**get_sp**()

**Return type**
int

**get_stack_address**(*offset*)

**Return type**
int | None

**Parameters**
**offset** (*Base*)

**property environment**

**property dep_graph**

**copy**(*discard_tmpdefs=False*)

**Return type**
Self

**merge**(**others*)

**Return type**
tuple[Self, bool]

**Parameters**
**others** (*Self*)

**compare**(*other*)

**Return type**
bool

**Parameters**
**other** (ReachingDefinitionsState)

**move_codelocs**(*new_codeloc*)

**Return type**
None

**Parameters**
**new_codeloc** (CodeLocation)

**kill_definitions**(*atom*)

Overwrite existing definitions w.r.t 'atom' with a dummy definition instance. A dummy definition will not be removed during simplification.

**Return type**
None

**Parameters**
**atom** (Atom)

**kill_and_add_definition**(*atom*, *data*, *dummy=False*, *tags=None*, *endness=None*, *annotated=False*, *uses=None*, *override_codeloc=None*)

**Return type**
tuple[*MultiValues* | None, set[*Definition*[TypeVar(A, bound= *Atom*), TypeVar(CodeLoc, bound= CodeLocation | AILCodeLocation)]]]

**Parameters**

- **atom** (Atom)
- **data** (MultiValues)
- **tags** (*set[Tag] | None*)
- **annotated** (*bool*)
- **uses** (*set[Definition[A, CodeLoc]] | None*)
- **override_codeloc** (*CodeLocation | None*)

**add_use**(*atom*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (Atom)
- **expr** (*Any | None*)

**add_use_by_def**(*definition*, *expr=None*)

**Return type**
None

**Parameters**

- **definition** (Definition[*A*, *CodeLoc*])
- **expr** (*Any | None*)

**add_tmp_use**(*tmp*, *expr=None*)

**Return type**
None

**Parameters**

- **tmp** (*int*)
- **expr** (*Any | None*)

**add_tmp_use_by_defs**(*defs*, *expr=None*)

**Return type**
None

**Parameters**

- **defs** (*Iterable*[Definition[*A*, *CodeLoc*]])
- **expr** (*Any | None*)

**add_register_use**(*reg_offset*, *size*, *expr=None*)

**Return type**
None

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)
- **expr** (*Any | None*)

**add_register_use_by_defs**(*defs*, *expr=None*)

**Return type**
None

**Parameters**

- **defs** (*Iterable*[Definition[*A*, *CodeLoc*]])
- **expr** (*Any | None*)

**add_stack_use**(*stack_offset*, *size*, *expr=None*)

**Return type**
None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **expr** (*Any | None*)

**add_stack_use_by_defs**(*defs*, *expr=None*)

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**add_heap_use**(*heap_offset*, *size*, *expr=None*)

**Return type**
None

**Parameters**

- **heap_offset** (*int*)
- **size** (*int*)
- **expr** (*Any | None*)

**add_heap_use_by_defs**(*defs*, *expr=None*)

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**add_memory_use_by_def**(*definition*, *expr=None*)

**Parameters**

- **definition** (*Definition[A, CodeLoc]*)
- **expr** (*Any | None*)

**add_memory_use_by_defs**(*defs*, *expr=None*)

**Parameters**

- **defs** (*Iterable[Definition[A, CodeLoc]]*)
- **expr** (*Any | None*)

**get_definitions**(*atom*)

**Return type**
set[*Definition*[*Atom*, TypeVar(CodeLoc, bound= CodeLocation | AILCodeLocation)]]

**Parameters**
**atom** (*Atom | Definition[Atom, CodeLoc] | Iterable[Atom] | Iterable[Definition[Atom, CodeLoc]] | MultiValues*)

**get_values**(*spec*)

**Return type**
*MultiValues* | None

**Parameters**
**spec** (*A | Definition[A, CodeLoc] | Iterable[A]*)

**get_one_value**(*spec*, *strip_annotations=False*)

**Return type**
BV | None

**Parameters**

- **spec** (*A | Definition[A, CodeLoc] | Iterable[A] | Iterable[Definition[A, CodeLoc]]*)
- **strip_annotations** (*bool*)

**get_concrete_value**(*spec*, *cast_to=<class 'int'>*)

**Overloads**

- **self**, **spec** (Atom | Definition[Atom, CodeLoc] | Iterable[Atom]), **cast_to** (type[int]) → int | None
- **self**, **spec** (Atom | Definition[Atom, CodeLoc] | Iterable[Atom]), **cast_to** (type[bytes]) → bytes | None

**Parameters**

- **spec** (*Atom | Definition[Atom, CodeLoc] | Iterable[Atom]*)
- **cast_to** (*type[int | bytes]*)

**Return type**
int | *bytes* | None

**mark_guard**(*target*)

**mark_const**(*value*, *size*)

**Parameters**

- **value** (*int*)
- **size** (*int*)

**downsize**()

**deref**(*pointer*, *size*, *endness=Endness.BE*)

**Overloads**

- **self**, **pointer** (int | claripy.ast.BV | HeapAddress | SpOffset), **size** (int | DerefSize), **endness** (archinfo.Endness) → MemoryLocation | None
- **self**, **pointer** (MultiValues | A | Definition | Iterable[A] | Iterable[Definition[A, CodeLoc]]), **size** (int | DerefSize), **endness** (archinfo.Endness) → set[MemoryLocation]

**Parameters**

- **pointer** (*MultiValues[BV] | Atom | Definition[Atom, CodeLoc] | Iterable[Atom] | Iterable[Definition[Atom, CodeLoc]] | int | BV | HeapAddress | SpOffset*)
- **size** (*int | DerefSize*)
- **endness** (*Endness*)

### angr.analyses.reaching_definitions.reaching_definitions

**class** angr.analyses.reaching_definitions.reaching_definitions.**ReachingDefinitionsAnalysis**

Bases: *ForwardAnalysis*[*ReachingDefinitionsState*, NodeType, object, object, object], *Analysis*

ReachingDefinitionsAnalysis is a text-book implementation of a static data-flow analysis that works on either a function or a block. It supports both VEX and AIL. By registering observers to observation points, users may use this analysis to generate use-def chains, def-use chains, and reaching definitions, and perform other traditional data-flow analyses such as liveness analysis.

- I've always wanted to find a better name for this analysis. Now I gave up and decided to live with this name for the foreseeable future (until a better name is proposed by someone else).
- Aliasing is definitely a problem, and I forgot how aliasing is resolved in this implementation. I'll leave this as a post-graduation TODO.
- Some more documentation and examples would be nice.

**__init__**(*subject*, *func_graph=None*, *max_iterations=30*, *track_tmps=False*, *track_consts=True*, *observation_points=None*, *init_state=None*, *init_context=None*, *state_initializer=None*, *cc=None*, *function_handler=None*, *observe_all=False*, *visited_blocks=None*, *dep_graph=True*, *observe_callback=None*, *canonical_size=8*, *stack_pointer_tracker=None*, *use_callee_saved_regs_at_return=True*, *interfunction_level=0*, *track_liveness=True*, *func_addr=None*, *element_limit=5*, *merge_into_tops=True*, *variable_map=None*)

**Parameters**

- **subject** (*Subject* | Block | *Block* | *Function* | str) – The subject of the analysis: a function, or a single basic block
- **func_graph** – Alternative graph for function.graph.
- **max_iterations** – The maximum number of iterations before the analysis is terminated.
- **track_tmps** – Whether or not temporary variables should be taken into consideration during the analysis.
- **observation_points** (Iterable[tuple[Literal['insn', 'node', 'stmt', 'exit'], int | tuple[int, int] | tuple[int, int, int], *ObservationPointType*]] | None) – A collection of tuples of ("node"|"insn", ins_addr, OP_TYPE) defining where reaching definitions should be copied and stored. OP_TYPE can be OP_BEFORE or OP_AFTER.
- **init_state** (*ReachingDefinitionsState* | None) – An optional initialization state. The analysis creates and works on a copy. Default to None: the analysis then initialize its own abstract state, based on the given <Subject>.
- **init_context** – If init_state is not given, this is used to initialize the context field of the initial state's CodeLocation. The only default-supported type which may go here is a tuple of integers, i.e. a callstack. Anything else requires a custom FunctionHandler.
- **cc** – Calling convention of the function.
- **function_handler** (*FunctionHandler* | None) – The function handler to update the analysis state and results on function calls.
- **observe_all** – Observe every statement, both before and after.
- **visited_blocks** – A set of previously visited blocks.
- **dep_graph** (*DepGraph* | bool | None) – An initial dependency graph to add the result of the analysis to. Set it to None to skip dependency graph generation.

- **canonical_size** – The sizes (in bytes) that objects with an UNKNOWN_SIZE are treated as for operations where sizes are necessary.
- **dep_graph** – Set this to True to generate a dependency graph for the subject. It will be available as *result.dep_graph*.
- **interfunction_level** (int) – The number of functions we should recurse into. This parameter is only used if function_handler is not provided.
- **track_liveness** (bool) – Whether to track liveness information. This can consume sizeable amounts of RAM on large functions. (e.g. ~15GB for a function with 4k nodes)
- **merge_into_tops** (bool) – Merge known values into TOP if TOP is present. If True: {TOP} V {0xabc} = {TOP} If False: {TOP} V {0xabc} = {TOP, 0xabc}
- **state_initializer** (RDAStateInitializer | None)
- **func_addr** (*int* | None)
- **element_limit** (*int*)

**property observed_results:** dict[tuple[str, int, int], *LiveDefinitions*]

**property all_definitions**

**property all_uses**

**property one_result**

**property dep_graph:** *DepGraph*

**property visited_blocks**

**get_reaching_definitions_by_insn**(*ins_addr*, *op_type*)

**get_reaching_definitions_by_node**(*node_addr*, *op_type*)

**node_observe**(*node_addr*, *state*, *op_type*, *node_idx=None*)

**Parameters**

- **node_addr** (int) – Address of the node.
- **state** (*ReachingDefinitionsState*) – The analysis state.
- **op_type** (*ObservationPointType*) – Type of the observation point. Must be one of the following: OP_BEFORE, OP_AFTER.
- **node_idx** (int | None) – ID of the node. Used in AIL to differentiate blocks with the same address.

**Return type**
None

**insn_observe**(*insn_addr*, *stmt*, *block*, *state*, *op_type*)

**Parameters**

- **insn_addr** (int) – Address of the instruction.
- **stmt** (*Statement* | IRStmt) – The statement.
- **block** (*Block* | Block) – The current block.
- **state** (*ReachingDefinitionsState*) – The abstract analysis state.

- **op_type** (*ObservationPointType*) – Type of the observation point. Must be one of the following: OP_BEORE, OP_AFTER.

**Return type**
None

**stmt_observe**(*stmt_idx*, *stmt*, *block*, *state*, *op_type*)

**Parameters**

- **stmt_idx** (int)
- **stmt** (*Statement* | IRStmt)
- **block** (*Block* | Block)
- **state** (*ReachingDefinitionsState*)
- **op_type** (*ObservationPointType*)

**Return type**
None

**Returns**

**exit_observe**(*node_addr*, *exit_stmt_idx*, *block*, *state*, *node_idx=None*)

**Parameters**

- **node_addr** (*int*)
- **exit_stmt_idx** (*int*)
- **block** (Block | *Block*)
- **state** (ReachingDefinitionsState)
- **node_idx** (*int* | *None*)

**property subject**

**callsites_to**(*target*)

**Return type**
Iterable[*FunctionCallRelationships*]

**Parameters**
**target** (*int* | *str* | Function)

### angr.analyses.reaching_definitions.subject

**class** angr.analyses.reaching_definitions.subject.**SubjectType**

Bases: Enum

**Function = 1**

**Block = 2**

**CallTrace = 3**

**class** angr.analyses.reaching_definitions.subject.**Subject**

Bases: object

**__init__**(*content*, *func_graph=None*, *cc=None*)

The thing being analysed, and the way (visitor) to analyse it.

**Parameters**

- **content** (*Union[ailment.Block, angr.Block, Function]*) – Thing to be analysed.
- **func_graph** (*networkx.DiGraph*) – Alternative graph for function.graph.
- **cc** (SimCC) – Calling convention of the function.

**property cc**

**property content**

**property func_graph**

**property type**

**property visitor: *FunctionGraphVisitor* | *SingleNodeGraphVisitor***

### 10.3.41 angr.analyses.reassembler

**exception** angr.analyses.reassembler.**BinaryError**

Bases: Exception

**exception** angr.analyses.reassembler.**InstructionError**

Bases: *BinaryError*

**exception** angr.analyses.reassembler.**ReassemblerFailureNotice**

Bases: *BinaryError*

angr.analyses.reassembler.**string_escape**(*s*)

angr.analyses.reassembler.**fill_reg_map**()

angr.analyses.reassembler.**split_operands**(*s*)

angr.analyses.reassembler.**is_hex**(*s*)

**class** angr.analyses.reassembler.**Label**

Bases: object

**g_label_ctr = count(0)**

**__init__**(*binary*, *name*, *original_addr=None*)

**property operand_str**

**property offset**

**static new_label**(*binary*, *name=None*, *function_name=None*, *original_addr=None*, *data_label=False*)

**class** angr.analyses.reassembler.**DataLabel**

Bases: *Label*

**__init__**(*binary*, *original_addr*, *name=None*)

**property operand_str**

class angr.analyses.reassembler.**FunctionLabel**

Bases: *Label*

**__init__**(*binary*, *function_name*, *original_addr*, *plt=False*)

**property function_name**

**property operand_str**

class angr.analyses.reassembler.**ObjectLabel**

Bases: *Label*

**__init__**(*binary*, *symbol_name*, *original_addr*, *plt=False*)

**property symbol_name**

**property operand_str**

class angr.analyses.reassembler.**NotypeLabel**

Bases: *Label*

**__init__**(*binary*, *symbol_name*, *original_addr*, *plt=False*)

**property symbol_name**

**property operand_str**

class angr.analyses.reassembler.**SymbolManager**

Bases: object

SymbolManager manages all symbols in the binary.

**__init__**(*binary*, *cfg*)

Constructor.

**Parameters**

- **binary** (Reassembler) – The Binary analysis instance.
- **cfg** (*angr.analyses.CFG*) – The CFG analysis instance.

**Returns**

None

**get_unique_symbol_name**(*symbol_name*)

**new_label**(*addr*, *name=None*, *is_function=None*, *force=False*)

**label_got**(*addr*, *label*)

Mark a certain label as assigned (to an instruction or a block of data).

**Parameters**

- **addr** (*int*) – The address of the label.
- **label** (angr.analyses.reassembler.Label) – The label that is just assigned.

**Returns**

None

class angr.analyses.reassembler.**Operand**

Bases: object

**__init__**(*binary*, *insn_addr*, *insn_size*, *capstone_operand*, *operand_str*, *mnemonic*, *operand_offset*, *syntax=None*)

Constructor.

**Parameters**

- **binary** (Reassembler) – The Binary analysis.
- **insn_addr** (*int*) – Address of the instruction.
- **capstone_operand**
- **operand_str** (*str*) – the string representation of this operand
- **mnemonic** (*str*) – Mnemonic of the instruction that this operand belongs to.
- **operand_offset** (*int*) – offset of the operand into the instruction.
- **syntax** (*str*) – Provide a way to override the default syntax coming from *binary*.

**Returns**
None

**assembly**()

**property is_immediate**

**property symbolized**

**class** angr.analyses.reassembler.**Instruction**

Bases: object

High-level representation of an instruction in the binary

**__init__**(*binary*, *addr*, *size*, *insn_bytes*, *capstone_instr*)

**Parameters**

- **binary** (Reassembler) – The Binary analysis
- **addr** (*int*) – Address of the instruction
- **size** (*int*) – Size of the instruction
- **insn_bytes** (*str*) – Instruction bytes
- **capstone_instr** – Capstone Instr object.

**Returns**
None

**assign_labels**()

**dbg_comments**()

**assembly**(*comments=False*, *symbolized=True*)

**Returns**

**class** angr.analyses.reassembler.**BasicBlock**

Bases: object

BasicBlock represents a basic block in the binary.

**__init__**(*binary*, *addr*, *size*, *x86_getpc_retsite=False*)

Constructor.

**Parameters**

- **binary** (Reassembler) – The Binary analysis.
- **addr** (*int*) – Address of the block
- **size** (*int*) – Size of the block
- **x86_getpc_retsite** (*bool*)

**Returns**
None

**assign_labels**()

**assembly**(*comments=False*, *symbolized=True*)

**instruction_addresses**()

**class** angr.analyses.reassembler.**Procedure**

Bases: object

Procedure in the binary.

**__init__**(*binary*, *function=None*, *addr=None*, *size=None*, *name=None*, *section='.text'*, *asm_code=None*)

Constructor.

**Parameters**

- **binary** (Reassembler) – The Binary analysis.
- **function** (*angr.knowledge.Function*) – The function it represents
- **addr** (*int*) – Address of the function. Not required if *function* is provided.
- **size** (*int*) – Size of the function. Not required if *function* is provided.
- **section** (*str*) – Which section this function comes from.

**Returns**
None

**property name**

Get function name from the labels of the very first block. :return: Function name if there is any, None otherwise :rtype: string

**property is_plt**

If this function is a PLT entry or not. :return: True if this function is a PLT entry, False otherwise :rtype: bool

**assign_labels**()

**assembly**(*comments=False*, *symbolized=True*)

Get the assembly manifest of the procedure.

**Parameters**

- **comments**
- **symbolized**

**Returns**
A list of tuples (address, basic block assembly), ordered by basic block addresses

**Return type**
list

`instruction_addresses()`

Get all instruction addresses in the binary.

**Returns**
A list of sorted instruction addresses.

**Return type**
list

`class angr.analyses.reassembler.ProcedureChunk`

Bases: *Procedure*

Procedure chunk.

`__init__(project, addr, size)`

Constructor.

**Parameters**

- **project**
- **addr**
- **size**

**Returns**

`class angr.analyses.reassembler.Data`

Bases: `object`

`__init__(binary, memory_data=None, section=None, section_name=None, name=None, size=None, sort=None, addr=None, initial_content=None)`

`property content`

`shrink(new_size)`

Reduce the size of this block

**Parameters**
**new_size** (*int*) – The new size

**Returns**
None

`desymbolize()`

We believe this was a pointer and symbolized it before. Now we want to desymbolize it.

The following actions are performed: - Reload content from memory - Mark the sort as 'unknown'

**Returns**
None

`assign_labels()`

`assembly(comments=False, symbolized=True)`

class angr.analyses.reassembler.**Relocation**

Bases: object

**__init__**(*addr*, *ref_addr*, *sort*)

class angr.analyses.reassembler.**Reassembler**

Bases: *Analysis*

High-level representation of a binary with a linear representation of all instructions and data regions. After calling "symbolize", it essentially acts as a binary reassembler.

Tested on CGC, x86 and x86-64 binaries.

Disclaimer: The reassembler is an empirical solution. Don't be surprised if it does not work on some binaries.

**__init__**(*syntax='intel'*, *remove_cgc_attachments=True*, *log_relocations=True*)

**property instructions**

Get a list of all instructions in the binary

**Returns**
A list of (address, instruction)

**Return type**
tuple

**property relocations**

**property inserted_asm_before_label**

**property inserted_asm_after_label**

**property main_executable_regions**

return:

**property main_nonexecutable_regions**

return:

**disassemble_block**(*block*)

Disassemble a block using the assembly syntax configured for this Reassembler.

When AT&T syntax is requested, a capstone instance dedicated to this analysis is used, so that neither the project-wide Arch object nor the capstone instance cached on it is modified.

**Parameters**
**block** – The angr Block to disassemble.

**Returns**
A CapstoneBlock instance.

**Return type**
*CapstoneBlock*

**section_alignment**(*section_name*)

Get the alignment for the specific section. If the section is not found, 16 is used as default.

**Parameters**
**section_name** (*str*) – The section.

**Returns**
The alignment in bytes.

**Return type**
int

`main_executable_regions_contain`(*addr*)

**Parameters**
**addr**

**Returns**

`main_executable_region_limbos_contain`(*addr*)

Sometimes there exists a pointer that points to a few bytes before the beginning of a section, or a few bytes after the beginning of the section. We take care of that here.

**Parameters**
**addr** (*int*) – The address to check.

**Returns**
A 2-tuple of (bool, the closest base address)

**Return type**
tuple

`main_nonexecutable_regions_contain`(*addr*)

**Parameters**
**addr** (*int*) – The address to check.

**Returns**
True if the address is inside a non-executable region, False otherwise.

**Return type**
bool

`main_nonexecutable_region_limbos_contain`(*addr*, *tolerance_before=64*, *tolerance_after=64*)

Sometimes there exists a pointer that points to a few bytes before the beginning of a section, or a few bytes after the beginning of the section. We take care of that here.

**Parameters**
**addr** (*int*) – The address to check.

**Returns**
A 2-tuple of (bool, the closest base address)

**Return type**
tuple

`register_instruction_reference`(*insn_addr*, *ref_addr*, *sort*, *operand_offset*)

`register_data_reference`(*data_addr*, *ref_addr*)

`add_label`(*name*, *addr*)

Add a new label to the symbol manager.

**Parameters**

- **name** (*str*) – Name of the label.
- **addr** (*int*) – Address of the label.

**Returns**
None

**insert_asm**(*addr*, *asm_code*, *before_label=False*)

Insert some assembly code at the specific address. There must be an instruction starting at that address.

**Parameters**

- **addr** (*int*) – Address of insertion
- **asm_code** (*str*) – The assembly code to insert

**Returns**
None

**append_procedure**(*name*, *asm_code*)

Add a new procedure with specific name and assembly code.

**Parameters**

- **name** (*str*) – The name of the new procedure.
- **asm_code** (*str*) – The assembly code of the procedure

**Returns**
None

**append_data**(*name*, *initial_content*, *size*, *readonly=False*, *sort='unknown'*)

Append a new data entry into the binary with specific name, content, and size.

**Parameters**

- **name** (*str*) – Name of the data entry. Will be used as the label.
- **initial_content** (bytes) – The initial content of the data entry.
- **size** (*int*) – Size of the data entry.
- **readonly** (*bool*) – If the data entry belongs to the readonly region.
- **sort** (*str*) – Type of the data.

**Returns**
None

**remove_instruction**(*ins_addr*)

**Parameters**
**ins_addr**

**Returns**

**randomize_procedures**()

**Returns**

**symbolize**()

**assembly**(*comments=False*, *symbolized=True*)

**remove_cgc_attachments**()

Remove CGC attachments.

**Returns**
True if CGC attachments are found and removed, False otherwise

**Return type**
bool

**remove_unnecessary_stuff**()

Remove unnecessary functions and data

**Returns**
None

**remove_unnecessary_stuff_glibc**()

**fast_memory_load**(*addr*, *size*, *data_type*, *endness='Iend_LE'*)

Load memory bytes from loader's memory backend.

**Parameters**

- **addr** (*int*) – The address to begin memory loading.
- **size** (*int*) – Size in bytes.
- **data_type** – Type of the data.
- **endness** (*str*) – Endianness of this memory load.

**Returns**
Data read out of the memory.

**Return type**
int or *bytes* or str or None

### 10.3.42 angr.analyses.s_liveness

**class** angr.analyses.s_liveness.**SLivenessModel**

Bases: object

The SLiveness model that stores LiveIn and LiveOut sets for each block in a partial-SSA function.

Blocks are identified by address and (block) index.

**__init__**()

**class** angr.analyses.s_liveness.**SLivenessAnalysis**

Bases: *Analysis*

Calculates LiveIn and LiveOut sets for each block in a partial-SSA function.

**__init__**(*func*, *func_graph*, *entry=None*, *func_addr=None*, *arg_vvars=None*)

**interference_graph**()

Generate an interference graph based on the liveness analysis result.

**Returns**
A networkx.Graph instance.

**live_vars_by_stmt**()

Get a mapping from statements to live variables at the point of the statement.

**Return type**
defaultdict[tuple[int, int | None], dict[int, set[int]]]

**Returns**
A dictionary mapping statements to sets of live variable IDs.

### 10.3.43 angr.analyses.s_propagator

angr.analyses.s_propagator.**sprop_cache_scope**(*cache*)

Install `cache` as the active AILBlockWalker cache for the current thread.

**Parameters**
**cache** (*dict[tuple[type, ...], AILWhitelistExprTypeWalker]*)

**class** angr.analyses.s_propagator.**SPropagatorModel**

Bases: `object`

The SPropagator model that stores replacements for virtual variables.

**__init__**()

**class** angr.analyses.s_propagator.**SPropagator**

Bases: `object`

Constant and expression propagation that only supports SSA AIL graphs.

Deliberately not an `Analysis` for better speed.

**__init__**(*project*, *subject*, ***, *ail_manager*, *func_graph=None*, *only_consts=True*, *stack_pointer_tracker=None*, *func_args=None*, *func_addr=None*, *stack_arg_offsets=None*)

**Parameters**

- **project** (*Project*)
- **subject** (*Block | Function*)
- **ail_manager** (*Manager*)
- **func_graph** (*DiGraph | None*)
- **only_consts** (*bool*)
- **func_args** (*set[VirtualVariable] | None*)
- **func_addr** (*int | None*)
- **stack_arg_offsets** (*set[int] | None*)

**property replacements**

**property dead_vvar_ids**

**static is_global_variable_updated**(*func_graph*, *block_dict*, *varid*, *gv_addr*, *gv_size*, *defloc*, *useloc*)

**Return type**
bool

**Parameters**

- **varid** (*int*)
- **gv_addr** (*int*)
- **gv_size** (*int*)
- **defloc** (*AILCodeLocation*)
- **useloc** (*AILCodeLocation*)

**static is_vvar_used_for_addr_loading_switch_case**(*uselocs*, *blocks*)

Check if a virtual variable is used for loading an address in a switch-case construct.

**Parameters**

- **uselocs** (set[*AILCodeLocation*]) – The use locations of the virtual variable.
- **blocks** – All blocks of the current function.

**Return type**
bool

**Returns**
True if the virtual variable is used for loading an address in a switch-case construct, False otherwise.

**static replace**(*replacements*, *loc*, *expr*, *value*)

**Return type**
None

**Parameters**

- **replacements** (*dict[*AILCodeLocation, *dict[*VirtualVariable | Tmp, Expression*]]*)
- **loc** (AILCodeLocation)
- **expr** (VirtualVariable | Tmp)
- **value** (Expression)

**class** angr.analyses.s_propagator.**SPropagatorAnalysis**

Bases: *Analysis*, *SPropagator*

A wrapper around SPropagator to make it an Analysis.

**__init__**(**args*, ***kwargs*)

### 10.3.44 angr.analyses.s_reaching_definitions

**class** angr.analyses.s_reaching_definitions.**SRDAModel**

Bases: object

The model for SRDA.

**__init__**(*func_graph*, *func_args*, *arch*, *platform=None*, *language=None*, *variable_map=None*, *functions=None*)

**Parameters**

- **platform** (*str* | *None*)
- **language** (*str* | *None*)
- **functions** (FunctionManager | *None*)

**add_vvar_use**(*vvar_id*, *expr*, *loc*)

**Return type**
None

**Parameters**

- **vvar_id** (*int*)

- **expr** (VirtualVariable | *None*)
- **loc** (AILCodeLocation)

**add_vvar_def**(*vvar_id*, *loc*)

Record the definition of a vvar id, keeping all_vvar_definitions and vvar_defs_by_loc in sync.

**Return type**
None

**Parameters**

- **vvar_id** (*int*)
- **loc** (AILCodeLocation)

**remove_vvar_def**(*vvar_id*)

Drop the definition of a vvar id from both all_vvar_definitions and vvar_defs_by_loc.

**Return type**
None

**Parameters**
**vvar_id** (*int*)

**update_after_block_edits**(*edited_blocks*)

Incrementally update the model after the statements of `edited_blocks` were edited *in place* (statement indices preserved, e.g. by replacing removed statements with `NoOp` placeholders). vvar definitions and explicit vvar uses inside the edited blocks are recomputed from the edited blocks; implicit uses (`expr is None`, e.g. call-site argument registers) and all unedited blocks are left untouched.

This is designed to be equivalent to a full SRDA rebuild on the edited graph for the kind of edits performed by dead-assignment removal: statements are only removed (turned into NoOp) or rewritten in place, never inserted or reordered, and removed vvars are dead (so they are never used elsewhere, including by phi nodes or implicit call-site uses). Tmp tracking is not updated (AILSimplifier does not track tmps).

**Return type**
None

**canonical_form**()

An order-insensitive snapshot of the model's vvar-keyed data, for asserting equivalence between an incrementally-updated model and a freshly-rebuilt one (used by the incremental-update verification harness).

**property all_definitions:** Iterator[*Definition*[*VirtualVariable*, *AILCodeLocation*]]

**get_definition_by_varid**(*varid*)

The definition of a vvar id, or None if it has none.

**Return type**
Optional[*Definition*[*VirtualVariable*, *AILCodeLocation*]]

**Parameters**
**varid** (*int*)

**is_phi_vvar_id**(*idx*)

**Return type**
bool

**Parameters**
**idx** (*int*)

**get_all_definitions**(*block_loc*)

**Return type**
set[*Definition*[*VirtualVariable* | *Tmp*, *AILCodeLocation*]]

**Parameters**
**block_loc** (AILCodeLocation)

**get_all_tmp_definitions**(*block_loc*)

**Return type**
set[*Definition*[*Tmp*, *AILCodeLocation*]]

**Parameters**
**block_loc** (*Address*)

**get_uses_by_location**(*loc*, *exprs=False*)

Retrieve all definitions that are used at a given location.

**Overloads**

- **self**, **loc** (AILCodeLocation), **exprs** (Literal[True]) → set[tuple[Definition[atoms.VirtualVariable, AILCodeLocation], Any | None]]
- **self**, **loc** (AILCodeLocation), **exprs** (Literal[False]) → set[Definition[atoms.VirtualVariable, AILCodeLocation]]

**Parameters**

- **loc** (*AILCodeLocation*) – The code location.
- **exprs** (*bool*)

**Returns**
A set of definitions that are used at the given location.

**Return type**
set[Definition[VirtualVariable, AILCodeLocation]] | set[tuple[Definition[VirtualVariable, AILCodeLocation], *Any* | None]]

**get_defs_by_location**(*loc*)

Retrieve all vvar definitions at a given location.

**Parameters**
**loc** (*AILCodeLocation*) – The code location.

**Return type**
set[*Definition*[*VirtualVariable*, *AILCodeLocation*]]

**Returns**
A set of definitions that are defined at the given location.

**get_vvar_uses**(*obj*)

**Return type**
set[*AILCodeLocation*]

**Parameters**
**obj** (VirtualVariable | VirtualVariable)

**get_vvar_uses_with_expr**(*obj*)

**Return type**
set[tuple[*VirtualVariable* | None, *AILCodeLocation*]]

**Parameters**
**obj** (VirtualVariable | VirtualVariable)

**get_tmp_uses**(*obj*, *block_loc*)

**Return type**
set[*AILCodeLocation*]

**Parameters**

- **obj** (Tmp)
- **block_loc** (*Address*)

**get_uses_by_def**(*def_*)

**Return type**
set[*AILCodeLocation*]

**Parameters**
**def_** (Definition[VirtualVariable | Tmp, AILCodeLocation])

**class** angr.analyses.s_reaching_definitions.**SRDAView**

Bases: object

A view of SRDA model that provides various functionalities for querying the model.

**__init__**(*model*)

**Parameters**
**model** (SRDAModel)

**get_reg_vvar_by_stmt**(*reg_offset*, *min_size*, *block_addr*, *block_idx*, *stmt_idx*, *op_type*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **reg_offset** (*int*)
- **min_size** (*int*)
- **block_addr** (*int*)
- **block_idx** (*int* | *None*)
- **stmt_idx** (*int*)
- **op_type** (ObservationPointType)

**get_stack_vvar_by_stmt**(*stack_offset*, *size*, *block_addr*, *block_idx*, *stmt_idx*, *op_type*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **block_addr** (*int*)
- **block_idx** (*int* | *None*)
- **stmt_idx** (*int*)

- **op_type** (ObservationPointType)

**get_reg_vvar_by_insn**(*reg_offset*, *min_size*, *addr*, *op_type*, *block_idx=None*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **reg_offset** (*int*)
- **min_size** (*int*)
- **addr** (*int*)
- **op_type** (ObservationPointType)
- **block_idx** (*int | None*)

**get_stack_vvar_by_insn**(*stack_offset*, *size*, *addr*, *op_type*, *block_idx=None*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **addr** (*int*)
- **op_type** (ObservationPointType)
- **block_idx** (*int | None*)

**get_vvar_value**(*vvar*)

**Return type**
*Expression* | None

**Parameters**
**vvar** (VirtualVariable)

**observe**(*observation_points*, *entry=None*)

**Parameters**

- **observation_points** (*list[tuple[Literal['insn', 'node', 'stmt', 'exit'], int | tuple[int, int] | tuple[int, int, int], ~angr.knowledge_plugins.key_definitions.constants.ObservationPointType]]*)
- **entry** (*Block | None*)

**class** angr.analyses.s_reaching_definitions.**SReachingDefinitions**

Bases: object

Constant and expression propagation that only supports SSA AIL graphs.

Deliberately not an `Analysis`: it is instantiated hundreds of times per decompilation (often once per block), so it skips the analysis-factory ceremony. Instantiate it directly with the project as the first argument; exceptions always propagate.

**__init__**(*project*, *subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*, *variable_map=None*)

class angr.analyses.s_reaching_definitions.**SReachingDefinitionsAnalysis**

Bases: *Analysis*, *SReachingDefinitions*

A wrapper around SReachingDefinitions to make it usable as an `Analysis` and registered in the analysis hub.

**__init__**(*subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*, *variable_map=None*)

angr.analyses.s_reaching_definitions.**populate_model**(*model*, *blocks*, *func_args*, ***, *fix_undefined_vvars=True*, *track_tmps=False*)

Populate the scan-derived part of an SRDAModel (vvar/tmp definitions and uses, phi bookkeeping) with a linear scan over `blocks`. An SRDAModel is never serialized; it is always rebuilt from an AIL graph through this function (via `SReachingDefinitions` or directly).

**Return type**
None

**Parameters**

- **model** (*SRDAModel*)
- **blocks** (*dict[tuple[int, int | None], Block]*)
- **func_args** (*set[VirtualVariable] | None*)
- **fix_undefined_vvars** (*bool*)
- **track_tmps** (*bool*)

## Submodules

*s_rda_model*
*s_rda_view*
*s_reaching_definitions*

### angr.analyses.s_reaching_definitions.s_rda_model

class angr.analyses.s_reaching_definitions.s_rda_model.**SRDAModel**

Bases: `object`

The model for SRDA.

**__init__**(*func_graph*, *func_args*, *arch*, *platform=None*, *language=None*, *variable_map=None*, *functions=None*)

**Parameters**

- **platform** (*str | None*)
- **language** (*str | None*)
- **functions** (*FunctionManager | None*)

**add_vvar_use**(*vvar_id*, *expr*, *loc*)

**Return type**
None

**Parameters**

- **vvar_id** (*int*)

- **expr** (VirtualVariable | *None*)
- **loc** (AILCodeLocation)

**add_vvar_def**(*vvar_id*, *loc*)

Record the definition of a vvar id, keeping all_vvar_definitions and vvar_defs_by_loc in sync.

**Return type**
None

**Parameters**

- **vvar_id** (*int*)
- **loc** (AILCodeLocation)

**remove_vvar_def**(*vvar_id*)

Drop the definition of a vvar id from both all_vvar_definitions and vvar_defs_by_loc.

**Return type**
None

**Parameters**
**vvar_id** (*int*)

**update_after_block_edits**(*edited_blocks*)

Incrementally update the model after the statements of `edited_blocks` were edited *in place* (statement indices preserved, e.g. by replacing removed statements with `NoOp` placeholders). vvar definitions and explicit vvar uses inside the edited blocks are recomputed from the edited blocks; implicit uses (`expr is None`, e.g. call-site argument registers) and all unedited blocks are left untouched.

This is designed to be equivalent to a full SRDA rebuild on the edited graph for the kind of edits performed by dead-assignment removal: statements are only removed (turned into NoOp) or rewritten in place, never inserted or reordered, and removed vvars are dead (so they are never used elsewhere, including by phi nodes or implicit call-site uses). Tmp tracking is not updated (AILSimplifier does not track tmps).

**Return type**
None

**canonical_form**()

An order-insensitive snapshot of the model's vvar-keyed data, for asserting equivalence between an incrementally-updated model and a freshly-rebuilt one (used by the incremental-update verification harness).

**property all_definitions:** Iterator[*Definition*[*VirtualVariable*, *AILCodeLocation*]]

**get_definition_by_varid**(*varid*)

The definition of a vvar id, or None if it has none.

**Return type**
Optional[*Definition*[*VirtualVariable*, *AILCodeLocation*]]

**Parameters**
**varid** (*int*)

**is_phi_vvar_id**(*idx*)

**Return type**
bool

**Parameters**
**idx** (*int*)

**get_all_definitions**(*block_loc*)

**Return type**
set[*Definition*[*VirtualVariable* | *Tmp*, *AILCodeLocation*]]

**Parameters**
**block_loc** (AILCodeLocation)

**get_all_tmp_definitions**(*block_loc*)

**Return type**
set[*Definition*[*Tmp*, *AILCodeLocation*]]

**Parameters**
**block_loc** (*Address*)

**get_uses_by_location**(*loc*, *exprs=False*)

Retrieve all definitions that are used at a given location.

**Overloads**

- **self**, **loc** (AILCodeLocation), **exprs** (Literal[True]) → set[tuple[Definition[atoms.VirtualVariable, AILCodeLocation], Any | None]]
- **self**, **loc** (AILCodeLocation), **exprs** (Literal[False]) → set[Definition[atoms.VirtualVariable, AILCodeLocation]]

**Parameters**

- **loc** (*AILCodeLocation*) – The code location.
- **exprs** (*bool*)

**Returns**
A set of definitions that are used at the given location.

**Return type**
set[Definition[VirtualVariable, AILCodeLocation]] | set[tuple[Definition[VirtualVariable, AILCodeLocation], *Any* | None]]

**get_defs_by_location**(*loc*)

Retrieve all vvar definitions at a given location.

**Parameters**
**loc** (*AILCodeLocation*) – The code location.

**Return type**
set[*Definition*[*VirtualVariable*, *AILCodeLocation*]]

**Returns**
A set of definitions that are defined at the given location.

**get_vvar_uses**(*obj*)

**Return type**
set[*AILCodeLocation*]

**Parameters**
**obj** (VirtualVariable | VirtualVariable)

**get_vvar_uses_with_expr**(*obj*)

**Return type**
set[tuple[*VirtualVariable* | None, *AILCodeLocation*]]

**Parameters**
**obj** (VirtualVariable | VirtualVariable)

**get_tmp_uses**(*obj*, *block_loc*)

**Return type**
set[*AILCodeLocation*]

**Parameters**

- **obj** (Tmp)
- **block_loc** (*Address*)

**get_uses_by_def**(*def_*)

**Return type**
set[*AILCodeLocation*]

**Parameters**
**def_** (Definition[VirtualVariable | Tmp, AILCodeLocation])

angr.analyses.s_reaching_definitions.s_rda_model.**populate_model**(*model*, *blocks*, *func_args*, *, *fix_undefined_vvars=True*, *track_tmps=False*)

Populate the scan-derived part of an SRDAModel (vvar/tmp definitions and uses, phi bookkeeping) with a linear scan over `blocks`. An SRDAModel is never serialized; it is always rebuilt from an AIL graph through this function (via `SReachingDefinitions` or directly).

**Return type**
None

**Parameters**

- **model** (SRDAModel)
- **blocks** (*dict[tuple[int, int | None], Block]*)
- **func_args** (*set[VirtualVariable] | None*)
- **fix_undefined_vvars** (*bool*)
- **track_tmps** (*bool*)

### angr.analyses.s_reaching_definitions.s_rda_view

angr.analyses.s_reaching_definitions.s_rda_view.**get_call_clobbered_regs**(*call*, *variable_map*, *functions*, *arch*, *platform*, *language*)

**Return type**
set[int]

**Parameters**

- **call** (Call)
- **functions** (FunctionManager | *None*)
- **platform** (*str* | *None*)
- **language** (*str* | *None*)

class angr.analyses.s_reaching_definitions.s_rda_view.**RegVVarPredicate**

Bases: object

Implements a predicate that is used in get_reg_vvar_by_stmt_idx and get_reg_vvar_by_insn.

**__init__**(*reg_offset*, *min_size*, *vvars*, *arch*, *platform=None*, *language=None*, *variable_map=None*, *functions=None*)

**Parameters**

- **reg_offset** (*int*)
- **min_size** (*int*)
- **vvars** (*list[VirtualVariable]*)
- **platform** (*str | None*)
- **language** (*str | None*)
- **functions** (FunctionManager | *None*)

**predicate**(*stmt*)

**Return type**
bool

**Parameters**
**stmt** (Statement)

class angr.analyses.s_reaching_definitions.s_rda_view.**StackVVarPredicate**

Bases: object

Implements a predicate that is used in get_stack_vvar_by_stmt_idx and get_stack_vvar_by_insn.

**__init__**(*stack_offset*, *size*, *vvars*)

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **vvars** (*list[VirtualVariable]*)

**predicate**(*stmt*)

**Return type**
bool

**Parameters**
**stmt** (Statement)

class angr.analyses.s_reaching_definitions.s_rda_view.**SRDAView**

Bases: object

A view of SRDA model that provides various functionalities for querying the model.

**__init__**(*model*)

**Parameters**
**model** (SRDAModel)

get_reg_vvar_by_stmt(*reg_offset*, *min_size*, *block_addr*, *block_idx*, *stmt_idx*, *op_type*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **reg_offset** (*int*)
- **min_size** (*int*)
- **block_addr** (*int*)
- **block_idx** (*int | None*)
- **stmt_idx** (*int*)
- **op_type** (ObservationPointType)

get_stack_vvar_by_stmt(*stack_offset*, *size*, *block_addr*, *block_idx*, *stmt_idx*, *op_type*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **block_addr** (*int*)
- **block_idx** (*int | None*)
- **stmt_idx** (*int*)
- **op_type** (ObservationPointType)

get_reg_vvar_by_insn(*reg_offset*, *min_size*, *addr*, *op_type*, *block_idx=None*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **reg_offset** (*int*)
- **min_size** (*int*)
- **addr** (*int*)
- **op_type** (ObservationPointType)
- **block_idx** (*int | None*)

get_stack_vvar_by_insn(*stack_offset*, *size*, *addr*, *op_type*, *block_idx=None*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **addr** (*int*)
- **op_type** (ObservationPointType)

- **block_idx** (*int | None*)

**get_vvar_value**(*vvar*)

**Return type**
*Expression* | None

**Parameters**
**vvar** (VirtualVariable)

**observe**(*observation_points*, *entry=None*)

**Parameters**

- **observation_points** (*list[tuple[Literal['insn', 'node', 'stmt', 'exit'], int | tuple[int, int] | tuple[int, int, int], ~angr.knowledge_plugins.key_definitions.constants.ObservationPointType]]*)
- **entry** (*Block | None*)

### angr.analyses.s_reaching_definitions.s_reaching_definitions

**class** angr.analyses.s_reaching_definitions.s_reaching_definitions.**SReachingDefinitions**

Bases: object

Constant and expression propagation that only supports SSA AIL graphs.

Deliberately not an Analysis: it is instantiated hundreds of times per decompilation (often once per block), so it skips the analysis-factory ceremony. Instantiate it directly with the project as the first argument; exceptions always propagate.

**__init__**(*project*, *subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*, *variable_map=None*)

**class** angr.analyses.s_reaching_definitions.s_reaching_definitions.**SReachingDefinitionsAnalysis**

Bases: *Analysis*, *SReachingDefinitions*

A wrapper around SReachingDefinitions to make it usable as an Analysis and registered in the analysis hub.

**__init__**(*subject*, *func_addr=None*, *func_graph=None*, *func_args=None*, *use_callee_saved_regs_at_return=False*, *track_tmps=False*, *variable_map=None*)

## 10.3.45 angr.analyses.smc

**class** angr.analyses.smc.**TraceActions**

Bases: IntFlag

Describe memory access actions.

**WRITE = 1**

**EXECUTE = 2**

**__new__**(*value*)

**class** angr.analyses.smc.**TraceClassifier**

Bases: object

Classify traces.

**__init__**(*state=None*)

**Parameters**
**state** (SimState | *None*)

**act_mem_write**(*state*)

SimInspect callback for memory writes.

**Return type**
None

**act_instruction**(*state*)

SimInspect callback for instruction execution.

**Return type**
None

**instrument**(*state*)

Instrument *state* for tracing.

**Return type**
None

**get_smc_address_and_lengths**()

Evaluate the trace to find which areas of memory were both written to and executed.

**Return type**
Generator[tuple[int, int]]

**determine_smc**()

Evaluate the trace to find areas of memory that were both written to and executed.

**Return type**
bool

**pp**()

**class** angr.analyses.smc.**SelfModifyingCodeAnalysis**

Bases: *Analysis*

Determine if some piece of code is self-modifying.

This determination is made by simply executing. If an address is executed that is also written to, the code is determined to be self-modifying. The determination is stored in the *result* property. The *regions* property contains a list of (addr, length) regions that were both written to and executed.

**__init__**(*subject*, *max_bytes=0*, *state=None*)

**Parameters**

- **subject** (int | str | *Function* | None) – Subject of analysis
- **max_bytes** (int) – Maximum number of bytes from subject address. 0 for no limit (default).
- **state** (*SimState* | None) – State to begin executing from.

**regions:** list[tuple[int, int]]

**result:** bool

### 10.3.46 angr.analyses.soot_class_hierarchy

**exception** angr.analyses.soot_class_hierarchy.**SootClassHierarchyError**

Bases: Exception

**exception** angr.analyses.soot_class_hierarchy.**NoConcreteDispatch**

Bases: *SootClassHierarchyError*

**class** angr.analyses.soot_class_hierarchy.**SootClassHierarchy**

Bases: *Analysis*

Generate complete hierarchy.

**__init__**()

**init_hierarchy**()

**has_super_class**(*cls*)

**is_subclass_including**(*cls_child*, *cls_parent*)

**is_subclass**(*cls_child*, *cls_parent*)

**is_visible_method**(*cls*, *method*)

**is_visible_class**(*cls_from*, *cls_to*)

**get_super_classes**(*cls*)

**get_super_classes_including**(*cls*)

**get_implementers**(*interface*)

**get_sub_interfaces_including**(*interface*)

**get_sub_interfaces**(*interface*)

**get_sub_classes**(*cls*)

**get_sub_classes_including**(*cls*)

**resolve_abstract_dispatch**(*cls*, *method*)

**resolve_concrete_dispatch**(*cls*, *method*)

**resolve_special_dispatch**(*method*, *container*)

**resolve_invoke**(*invoke_expr*, *method*, *container*)

### 10.3.47 angr.analyses.stack_pointer_tracker

**class** angr.analyses.stack_pointer_tracker.**BottomType**

Bases: object

The bottom value for register values.

**class** angr.analyses.stack_pointer_tracker.**Constant**

Bases: object

Represents a constant value.

__init__(*val*)

val

class angr.analyses.stack_pointer_tracker.Register

Bases: object

Represent a register.

__init__(*offset*, *bitlen*)

offset

bitlen

class angr.analyses.stack_pointer_tracker.OffsetVal

Bases: object

Represent a value with an offset added.

__init__(*reg*, *offset*)

property reg

property offset

class angr.analyses.stack_pointer_tracker.Eq

Bases: object

Represent an equivalence condition.

__init__(*val0*, *val1*)

val0

val1

class angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerState

Bases: object

Abstract state for StackPointerTracker analysis with registers and memory values being in frozensets.

__init__(*regs*, *memory*, *is_tracking_memory*, *resilient*)

regs

memory

is_tracking_memory

resilient

unfreeze()

merge(*other*, *addr*, *reg_merge_cache*, *mem_merge_cache*)

**Parameters**

- **addr** (*int*)
- **reg_merge_cache** (*dict[tuple[int, int], Any]*)
- **mem_merge_cache** (*dict[tuple[int, int], Any]*)

class angr.analyses.stack_pointer_tracker.**StackPointerTrackerState**

Bases: object

Abstract state for StackPointerTracker analysis.

**__init__**(*regs*, *memory*, *is_tracking_memory*, *resilient*)

**Parameters**
**resilient** (*bool*)

**regs**

**memory**

**is_tracking_memory**

**resilient**

**give_up_on_memory_tracking**()

**store**(*addr*, *val*)

**pop_dirty**()

**load**(*addr*)

**get**(*reg*)

**put**(*reg*, *val*, *force=False*)

**Parameters**
**force** (*bool*)

**copy**()

**freeze**()

**merge**(*other*, *addr*, *reg_merge_cache*, *mem_merge_cache*)

**Parameters**

- **addr** (*int*)
- **reg_merge_cache** (*dict[tuple[int, int], Any]*)
- **mem_merge_cache** (*dict[tuple[int, int], Any]*)

exception angr.analyses.stack_pointer_tracker.**CouldNotResolveException**

Bases: Exception

An exception used in StackPointerTracker analysis to represent internal resolving failures.

class angr.analyses.stack_pointer_tracker.**StackPointerTracker**

Bases: *Analysis*, *ForwardAnalysis*

Track the offset of stack pointer at the end of each basic block of a function.

**reg_values:** dict[int, dict[int, Any]]

**mem_values:** dict[int, dict[int, Any]]

**reg_deltas:** dict[int, dict[int, dict[int, Any]]]

**mem_deltas:** dict[int, dict[int, dict[int, Any]]]

**offset_after**(*addr*, *reg*)

**offset_before**(*addr*, *reg*)

**offset_after_block**(*block_addr*, *reg*)

**offset_before_block**(*block_addr*, *reg*)

**constant_after**(*addr*, *reg*)

**constant_before**(*addr*, *reg*)

**constant_after_block**(*block_addr*, *reg*)

**constant_before_block**(*block_addr*, *reg*)

**property inconsistent**

**inconsistent_for**(*reg*)

**offsets_for**(*reg*)

### 10.3.48 angr.analyses.static_hooker

**class** angr.analyses.static_hooker.**StaticHooker**

Bases: *Analysis*

This analysis works on statically linked binaries - it finds the library functions statically linked into the binary and hooks them with the appropriate simprocedures.

Right now it only works on unstripped binaries, but hey! There's room to grow!

**__init__**(*library*, *binary=None*)

### 10.3.49 angr.analyses.typehoon

**class** angr.analyses.typehoon.**TypeTranslator**

Bases: object

Translate type constants to SimType and SimType to type constants.

**__init__**(*arch*, *func_addr=None*)

**Parameters**

- **arch** (*Arch*)
- **func_addr** (*int | None*)

**arch:** Arch

**func_addr:** int | None

**translated:** dict[*TypeConstant*, *SimType*]

**translated_simtypes:** dict[*SimType*, *TypeConstant*]

**structs**

**memo**: dict[str, *Struct*]

**named_struct_id_counter**

**struct_name_to_idx**

**known_structs**: dict[str, *SimStruct*]

**struct_name**()

Create a name for an auto-generated struct. Do not conflict with struct names in self.known_structs.

**Return type**
str

**tc2simtype**(*tc*)

**simtype2tc**(*simtype*)

**Return type**
*TypeConstant*

**Parameters**
**simtype** (SimType)

**lift**(*simtype*)

**Return type**
*TypeConstant*

**Parameters**
**simtype** (SimType)

**concretize**(*tc*)

**backpatch**(*st*, *translated*)

**Parameters**

- **st** (sim_type.SimType)
- **translated** (*dict*)

**Returns**

**class** angr.analyses.typehoon.**Typehoon**

Bases: *Analysis*

A spiritual tribute to the long-standing typehoon project that @jmg (John Grosen) worked on during his days in the angr team. Now I feel really bad of asking the poor guy to work directly on VEX IR without any fancy static analysis support as we have right now...

Typehoon analysis implements a pushdown system that simplifies and solves type constraints. Our type constraints are largely an implementation of the paper Polymorphic Type Inference for Machine Code by Noonan, Loginov, and Cok from GrammaTech (with missing functionality support and bugs, of course). Type constraints are collected by running VariableRecoveryFast (maybe VariableRecovery later as well) on a function, and then solved using this analysis.

User may specify ground truth, which will override all types at certain program points during constraint solving.

**__init__**(*constraints*, *func_var*, *ground_truth=None*, *var_mapping=None*, *must_struct=None*, *stackvar_max_sizes=None*, *stack_offset_tvs=None*, *constraint_set_degradation_threshold=150*, *type_translator=None*, *tv_manager=None*)

**Parameters**

- **constraints**
- **ground_truth** – A set of SimType-style solutions for some or all type variables. They will be respected during type solving.
- **var_mapping** (dict[*SimVariable*, set[*TypeVariable*]] | None)
- **must_struct** (set[*TypeVariable*] | None)
- **stackvar_max_sizes** (*dict[TypeVariable, int] | None*)
- **stack_offset_tvs** (*dict[int, TypeVariable] | None*)
- **constraint_set_degradation_threshold** (*int*)
- **type_translator** (*TypeTranslator | None*)
- **tv_manager** (*TypeVariableManager | None*)

**update_variable_types**(*func_addr*, *var_to_typevars*, *stack_offset_tvs=None*)

**Return type**
None

**Parameters**

- **func_addr** (*int | str*)
- **var_to_typevars** (*dict[SimVariable, set[TypeVariable]]*)
- **stack_offset_tvs** (*dict[int, TypeVariable] | None*)

**pp_constraints**()

Pretty-print constraints between *variables* using the variable mapping.

**Return type**
None

**pp_solution**()

Pretty-print solutions using the variable mapping.

**Return type**
None

## Submodules

| | |
|---|---|
| *dfa* | |
| *simple_solver* | |
| *translator* | |
| *typeconsts* | All type constants used in type inference. |
| *typehoon* | |
| *typevars* | |
| *variance* | |

### angr.analyses.typehoon.dfa

**exception** angr.analyses.typehoon.dfa.**EmptyEpsilonNFAError**

Bases: *AngrError*

A notification exception generated when the epsilon NFA is empty.

class angr.analyses.typehoon.dfa.**DFAConstraintSolver**

Bases: object

Implements a DFA-based graph solver.

**static graph_to_epsilon_nfa**(*graph*, *starts*, *ends*)

**Return type**
EpsilonNFA

**Parameters**

- **graph** (*DiGraph*)
- **starts** (*set*)
- **ends** (*set*)

**generate_constraints_between**(*graph*, *starts*, *ends*)

**Return type**
set

**Parameters**

- **graph** (*DiGraph*)
- **starts** (*set*)
- **ends** (*set*)

### angr.analyses.typehoon.simple_solver

class angr.analyses.typehoon.simple_solver.**TypeLattice**

Bases: object

A lattice of type constants with a lowest-common-ancestor table for caching purposes.

The lattice graph (*self.g*) should be treated as read-only, otherwise the cached LCA table will be invalidated once the lattice graph changes.

**__init__**(*g*)

**Parameters**
**g** (*DiGraph*)

**g: DiGraph**

**inverted**()

Return a new lattice with every edge reversed, i.e. ordered from the most specific to the most general.

**Return type**
*TypeLattice*

**lca**(*node_a*, *node_b*)

Return the lowest common ancestor of node_a and node_b, or None if they have none. Initialize the LCA table on the first query and memoize it for subsequent queries. This is critical for performance when the lattice is relatively small (< 100 nodes) and the number of queries is large.

Reconsider this implementation when the type lattice is large.

class angr.analyses.typehoon.simple_solver.**SketchNodeBase**

Bases: object

The base class for nodes in a sketch.

class angr.analyses.typehoon.simple_solver.**SketchNode**

Bases: *SketchNodeBase*

Represents a node in a sketch graph.

**__init__**(*typevar*)

**Parameters**
**typevar** (TypeVariable | DerivedTypeVariable)

**typevar:** *TypeVariable | DerivedTypeVariable*

**upper_bound:** *TypeConstant*

**lower_bound:** *TypeConstant*

**property size:** int | None

Best-effort estimation of the size of the typevar (in bits). Returns None if we cannot determine.

class angr.analyses.typehoon.simple_solver.**RecursiveRefNode**

Bases: *SketchNodeBase*

Represents a cycle in a sketch graph.

This is equivalent to sketches.LabelNode in the reference implementation of retypd.

**__init__**(*target*)

**Parameters**
**target** (TypeVariable | DerivedTypeVariable)

**target:** *TypeVariable | DerivedTypeVariable*

class angr.analyses.typehoon.simple_solver.**Sketch**

Bases: object

Describes the sketch of a type variable.

**__init__**(*solver*, *root*)

**Parameters**

- **solver** (SimpleSolver)
- **root** (TypeVariable)

**root:** *SketchNode*

**graph**

**node_mapping:** dict[*TypeVariable | DerivedTypeVariable*, *SketchNodeBase*]

**solver**

**lookup**(*typevar*)

**Return type**
*SketchNodeBase* | None

**Parameters**
**typevar** (TypeVariable | DerivedTypeVariable)

**add_edge**(*src*, *dst*, *label*)

**Return type**
None

**Parameters**

- **src** (SketchNodeBase)
- **dst** (SketchNodeBase)

**add_constraint**(*constraint*)

**Return type**
None

**Parameters**
**constraint** (TypeConstraint)

**static flatten_typevar**(*derived_typevar*)

**Return type**
tuple[*DerivedTypeVariable* | *TypeVariable* | *TypeConstant*, bool]

**Parameters**
**derived_typevar** (TypeVariable | TypeConstant | DerivedTypeVariable)

**class** angr.analyses.typehoon.simple_solver.**ConstraintGraphTag**

Bases: Enum

**__init__**(*n*)

**LEFT = 0**

**RIGHT = 1**

**UNKNOWN = 2**

**class** angr.analyses.typehoon.simple_solver.**FORGOTTEN**

Bases: Enum

**__init__**(*n*)

**PRE_FORGOTTEN = 0**

**POST_FORGOTTEN = 1**

**class** angr.analyses.typehoon.simple_solver.**ConstraintGraphNode**

Bases: object

**__init__**(*typevar*, *variance*, *tag*, *forgotten*)

**Parameters**

- **typevar** (TypeVariable | DerivedTypeVariable | TypeConstant)

- **variance** (Variance)
- **tag** (ConstraintGraphTag)
- **forgotten** (FORGOTTEN)

**typevar**

**variance**

**tag**

**forgotten**

**forget_last_label**()

**Return type**
tuple[*ConstraintGraphNode*, *BaseLabel*] | None

**recall**(*label*)

**Return type**
*ConstraintGraphNode*

**Parameters**
**label** (BaseLabel)

**inverse**()

**Return type**
*ConstraintGraphNode*

**inverse_wo_tag**()

Invert the variance only.

**Return type**
*ConstraintGraphNode*

angr.analyses.typehoon.simple_solver.**map_offsets_to_bases**(*candidate_bases*)

Map each candidate field offset to the offset of the lowest field that covers it.

`candidate_bases` maps a field offset to the set of sizes that this field is accessed with; a field at offset `o` covers the bytes in `[o, o + max(sizes))`. Fields may overlap, in which case the lowest one wins.

We only ever need the bases of offsets that are candidate bases themselves, so the covered ranges are compared directly instead of being expanded byte by byte – a single field can be hundreds of megabytes wide.

**Return type**
SortedDict

**Parameters**
**candidate_bases** (*SortedDict*)

**class** angr.analyses.typehoon.simple_solver.**SimpleSolver**

Bases: object

SimpleSolver is, by its name, a simple solver. Most of this solver is based on the (complex) simplification logic that the retypd paper describes and the retypd re-implementation (https://github.com/GrammaTech/retypd) implements. Additionally, we add some improvements to allow type propagation of known struct names, among a few other improvements.

__init__(*bits*, *constraints*, *typevars*, *constraint_set_degradation_threshold=150*, *stackvar_max_sizes=None*, *tv_manager=None*)

**Parameters**

- **bits** (*int*)
- **typevars** (*dict[TypeVariable, set[TypeVariable]]*)
- **constraint_set_degradation_threshold** (*int*)
- **stackvar_max_sizes** (*dict[TypeVariable, int] | None*)
- **tv_manager** (TypeVariableManager | *None*)

**processed_constraints_count:** int

**simplified_constraints_count:** int

**eqclass_constraints_count:** list[int]

get_tv_max_stack_size(*tv*)

Get the potential maximum stack variable size of a given type variable, if any. Also considers other type variables that are equivalent to the given type variable.

**Return type**
int | None

**Parameters**
**tv** (TypeVariable)

preprocess(*func_tv*)

**Parameters**
**func_tv** (TypeVariable)

solve()

Steps:

For each type variable, - Infer the shape in its sketch - Build the constraint graph - Collect all constraints - Apply constraints to derive the lower and upper bounds - Determine a solution for type variables with constraints - Rewrite the constraint graph by replacing determined type variables with their solutions - Solve repeatedly until all interesting type variables have solutions

By repeatedly solving until exhausting interesting type variables, we ensure the S-Trans rule is applied.

infer_shapes(*typevars*, *constraints*)

Computing sketches from constraint sets. Implements Algorithm E.1 in the retypd paper.

**Return type**
tuple[dict, dict[*TypeVariable*, *Sketch*]]

**Parameters**

- **typevars** (*set[TypeVariable]*)
- **constraints** (*set[TypeConstraint]*)

compute_quotient_graph(*constraints*)

Compute the quotient graph (the constraint graph modulo ~ in Algorithm E.1 in the retypd paper) with respect to a given set of type constraints.

**Parameters**
**constraints** (*set[TypeConstraint]*)

**join**(*t1*, *t2*)

**Return type**
*TypeConstant*

**Parameters**

- **t1** (TypeConstant)
- **t2** (TypeConstant)

**meet**(*t1*, *t2*)

**Return type**
*TypeConstant*

**Parameters**

- **t1** (TypeConstant)
- **t2** (TypeConstant)

**classmethod join_simtypes**(*t1*, *t2*, *arch*)

Compute the join (the least general common supertype) of two SimTypes on the type lattice.

The two types are lifted to internal type constants, joined on the base lattice for `arch.bits`, and the result is translated back into a SimType. When the two types have no meaningful common supertype, the result is a `SimTypeBottom`.

**Parameters**

- **t1** (*SimType*) – The first SimType.
- **t2** (*SimType*) – The second SimType.
- **arch** (Arch) – The architecture, used to determine the pointer size and to build the resulting SimType.

**Return type**
*SimType*

**Returns**
The join of `t1` and `t2` as a SimType.

**classmethod meet_simtypes**(*t1*, *t2*, *arch*)

Compute the meet (the greatest common subtype) of two SimTypes on the type lattice.

See *join_simtypes()* for details. When the two types have no meaningful common subtype, the result is a `SimTypeBottom`.

**Return type**
*SimType*

**Parameters**

- **t1** (SimType)
- **t2** (SimType)
- **arch** (*Arch*)

**static abstract**(*t*)

**Return type**
*TypeConstant* | *TypeVariable*

**Parameters**
**t** (TypeConstant | TypeVariable)

**determine**(*sketches*, *tvs*, *equivalence_classes*, *solution*, *nodes=None*)

Determine C-like types from sketches.

**Parameters**

- **sketches** – A dictionary storing sketches for each type variable.
- **solution** (dict) – The dictionary storing C-like types for each type variable. Output.
- **nodes** (set[*SketchNode*] | None) – Optional. Nodes that should be considered in the sketch.
- **equivalence_classes** (*dict[TypeVariable, TypeVariable]*)

**Return type**
None

**Returns**
None

**static dump_constraint_graph**(*graph*, *filename*)

Dump the constraint graph to a file.

**Return type**
None

**Parameters**

- **graph** (*DiGraph*)
- **filename** (*str*)

### angr.analyses.typehoon.translator

**class** angr.analyses.typehoon.translator.**SimTypeTempRef**

Bases: *SimType*

Represents a temporary reference to another type. TypeVariableReference is translated to SimTypeTempRef.

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Return type**
str

**Parameters**

- **indent** (*int | None*)
- **name_parens** (*bool*)

**class** angr.analyses.typehoon.translator.**TypeTranslator**

Bases: object

Translate type constants to SimType and SimType to type constants.

**__init__**(*arch*, *func_addr=None*)

**Parameters**

- **arch** (*Arch*)
- **func_addr** (*int | None*)

**arch:** *Arch*

**func_addr:** *int* | *None*

**translated:** *dict*[*TypeConstant*, *SimType*]

**translated_simtypes:** *dict*[*SimType*, *TypeConstant*]

**structs**

**memo:** *dict*[*str*, *Struct*]

**named_struct_id_counter**

**struct_name_to_idx**

**known_structs:** *dict*[*str*, *SimStruct*]

**struct_name**()

Create a name for an auto-generated struct. Do not conflict with struct names in self.known_structs.

**Return type**
*str*

**tc2simtype**(*tc*)

**simtype2tc**(*simtype*)

**Return type**
*TypeConstant*

**Parameters**
**simtype** (SimType)

**lift**(*simtype*)

**Return type**
*TypeConstant*

**Parameters**
**simtype** (SimType)

**concretize**(*tc*)

**backpatch**(*st*, *translated*)

**Parameters**

- **st** (sim_type.SimType)
- **translated** (*dict*)

**Returns**

### angr.analyses.typehoon.typeconsts

All type constants used in type inference. They can be mapped, translated, or rewritten to C-style types.

angr.analyses.typehoon.typeconsts.**memoize**(*f*)

**class** angr.analyses.typehoon.typeconsts.**TypeConstant**

Bases: object

SIZE = None

TYPE_HASH: int = 1984226046

__init__(*name=None*)

**Parameters**
**name** (*str | None*)

pp_str(*mapping*)

**Return type**
str

property size: int

replace(*mapping*, *memo=None*)

**Return type**
*TypeConstant*

**Parameters**

- **mapping** (*dict[int, TypeConstant]*)
- **memo** (*set[TypeConstant] | None*)

class angr.analyses.typehoon.typeconsts.**TopType**

Bases: *TypeConstant*

SIZE = 1

TYPE_HASH: int = 1194223761

class angr.analyses.typehoon.typeconsts.**BottomType**

Bases: *TypeConstant*

SIZE = 1

TYPE_HASH: int = 1602925492

class angr.analyses.typehoon.typeconsts.**Int**

Bases: *TypeConstant*

TYPE_HASH: int = 740072529

class angr.analyses.typehoon.typeconsts.**Int1**

Bases: *Int*

SIZE = 1

TYPE_HASH: int = 2677838409

class angr.analyses.typehoon.typeconsts.**Int8**

Bases: *Int*

SIZE = 1

TYPE_HASH: int = 3862964973

class angr.analyses.typehoon.typeconsts.**Int16**

Bases: *Int*

SIZE = 2

TYPE_HASH: int = 304553906

class angr.analyses.typehoon.typeconsts.Int32

Bases: *Int*

SIZE = 4

TYPE_HASH: int = 662485289

class angr.analyses.typehoon.typeconsts.Fd

Bases: *Int*

SIZE = 4

TYPE_HASH: int = 2904907581

class angr.analyses.typehoon.typeconsts.Int64

Bases: *Int*

SIZE = 8

TYPE_HASH: int = 3009996889

class angr.analyses.typehoon.typeconsts.SInt8

Bases: *Int8*

TYPE_HASH: int = 2123741175

class angr.analyses.typehoon.typeconsts.UInt8

Bases: *Int8*

TYPE_HASH: int = 4057285207

class angr.analyses.typehoon.typeconsts.SInt16

Bases: *Int16*

TYPE_HASH: int = 4024251221

class angr.analyses.typehoon.typeconsts.UInt16

Bases: *Int16*

TYPE_HASH: int = 965005384

class angr.analyses.typehoon.typeconsts.SInt32

Bases: *Int32*

TYPE_HASH: int = 3666253262

class angr.analyses.typehoon.typeconsts.UInt32

Bases: *Int32*

TYPE_HASH: int = 215971539

class angr.analyses.typehoon.typeconsts.SInt64

Bases: *Int64*

TYPE_HASH: int = 1318241470

class angr.analyses.typehoon.typeconsts.**UInt64**

Bases: *Int64*

**TYPE_HASH: int = 2563450787**

class angr.analyses.typehoon.typeconsts.**Int128**

Bases: *Int*

**SIZE = 16**

**TYPE_HASH: int = 3138480040**

class angr.analyses.typehoon.typeconsts.**Int256**

Bases: *Int*

**SIZE = 32**

**TYPE_HASH: int = 296641073**

class angr.analyses.typehoon.typeconsts.**Int512**

Bases: *Int*

**SIZE = 32**

**TYPE_HASH: int = 2011198889**

class angr.analyses.typehoon.typeconsts.**IntVar**

Bases: *Int*

**__init__**(*size*, *name=None*)

**Parameters**

**name** (*str | None*)

**property size: int**

**TYPE_HASH: int = 4044758036**

class angr.analyses.typehoon.typeconsts.**Float**

Bases: *TypeConstant*

**TYPE_HASH: int = 140800401**

class angr.analyses.typehoon.typeconsts.**Float32**

Bases: *Float*

**SIZE = 4**

**TYPE_HASH: int = 842912986**

class angr.analyses.typehoon.typeconsts.**Float64**

Bases: *Float*

**SIZE = 8**

**TYPE_HASH: int = 2787739050**

class angr.analyses.typehoon.typeconsts.**Pointer**

Bases: *TypeConstant*

__init__(*basetype*, *name=None*)

**Parameters**

- **basetype** (*TypeConstant | None*)
- **name** (*str | None*)

**basetype:** *TypeConstant* | None

**new**(*basetype*, *name=None*)

**Parameters**
**name** (*str | None*)

**replace**(*mapping*, *memo=None*)

**Return type**
*TypeConstant*

**Parameters**

- **mapping** (*dict[int, TypeConstant]*)
- **memo** (*set | None*)

**TYPE_HASH:** int = 4256780596

class angr.analyses.typehoon.typeconsts.**Pointer32**

Bases: *Pointer*, *Int32*

32-bit pointers.

__init__(*basetype=None*, *name=None*)

**Parameters**
**name** (*str | None*)

**TYPE_HASH:** int = 3659745635

class angr.analyses.typehoon.typeconsts.**Pointer64**

Bases: *Pointer*, *Int64*

64-bit pointers.

__init__(*basetype=None*, *name=None*)

**Parameters**
**name** (*str | None*)

**TYPE_HASH:** int = 1312233491

class angr.analyses.typehoon.typeconsts.**Array**

Bases: *TypeConstant*

__init__(*element=None*, *count=None*, *name=None*)

**Parameters**
**name** (*str | None*)

**element:** *TypeConstant* | None

**count:** int | None

property size: int

replace(*mapping*, *memo=None*)

Return type
*TypeConstant*

Parameters

- **mapping** (*dict[int, TypeConstant]*)
- **memo** (*set | None*)

TYPE_HASH: int = 1624097203

class angr.analyses.typehoon.typeconsts.Struct

Bases: *TypeConstant*

__init__(*fields=None*, *name=None*, *field_names=None*, *is_cppclass=False*, *idx=-1*)

Parameters

- **is_cppclass** (*bool*)
- **idx** (*int*)

property size: int

TYPE_HASH: int = 3175010840

class angr.analyses.typehoon.typeconsts.EnumVariant

Bases: object

__init__(*name*, *fields*, *discriminant*, *discriminant_size*, *size*)

fields: list[tuple[*TypeConstant*, str | None]]

class angr.analyses.typehoon.typeconsts.RustEnum

Bases: *TypeConstant*

__init__(*name=None*, *variants=None*)

get_variant(*name*)

Return type
*EnumVariant* | None

property size: int

replace(*mapping*, *memo=None*)

Return type
*TypeConstant*

Parameters

- **mapping** (*dict[int, TypeConstant]*)
- **memo** (*set | None*)

TYPE_HASH: int = 3260887885

class angr.analyses.typehoon.typeconsts.**Enum**

Bases: *TypeConstant*

Enum type constant for type inference.

**Variables**

- **members** (dict[str, int]) – Mapping of enum member names to their integer values
- **base_type** – The underlying type constant (defaults to Int32)
- **idx** – Unique identifier for this enum

**__init__**(*members=None*, *base_type=None*, *name=None*, *idx=-1*)

**Parameters**

- **members** (*dict[str, int] | None*)
- **base_type** (*TypeConstant | None*)
- **name** (*str | None*)
- **idx** (*int*)

**members: dict[str, int]**

**property size: int**

Return the size of the enum in bytes.

**replace**(*mapping*, *memo=None*)

**Return type**
*TypeConstant*

**Parameters**

- **mapping** (*dict[int, TypeConstant]*)
- **memo** (*set | None*)

**TYPE_HASH: int = 2928410991**

class angr.analyses.typehoon.typeconsts.**Function**

Bases: *TypeConstant*

**__init__**(*params*, *outputs*, *name=None*)

**Parameters**

- **params** (*list*)
- **outputs** (*list*)
- **name** (*str | None*)

**replace**(*mapping*, *memo=None*)

**Return type**
*TypeConstant*

**Parameters**

- **mapping** (*dict[int, TypeConstant]*)
- **memo** (*set | None*)

TYPE_HASH: int = 869823595

class angr.analyses.typehoon.typeconsts.TypeVariableReference

Bases: TypeConstant

__init__(*typevar*, *name=None*)

**Parameters**
**name** (*str | None*)

TYPE_HASH: int = 3866462522

angr.analyses.typehoon.typeconsts.int_type(*bits*)

**Return type**
*Int*

**Parameters**
**bits** (*int*)

angr.analyses.typehoon.typeconsts.signed_int_type(*bits*)

**Return type**
*Int*

**Parameters**
**bits** (*int*)

angr.analyses.typehoon.typeconsts.unsigned_int_type(*bits*)

**Return type**
*Int*

**Parameters**
**bits** (*int*)

angr.analyses.typehoon.typeconsts.float_type(*bits*)

**Return type**
*Float* | None

**Parameters**
**bits** (*int*)

### angr.analyses.typehoon.typehoon

class angr.analyses.typehoon.typehoon.Typehoon

Bases: Analysis

A spiritual tribute to the long-standing typehoon project that @jmg (John Grosen) worked on during his days in the angr team. Now I feel really bad of asking the poor guy to work directly on VEX IR without any fancy static analysis support as we have right now...

Typehoon analysis implements a pushdown system that simplifies and solves type constraints. Our type constraints are largely an implementation of the paper Polymorphic Type Inference for Machine Code by Noonan, Loginov, and Cok from GrammaTech (with missing functionality support and bugs, of course). Type constraints are collected by running VariableRecoveryFast (maybe VariableRecovery later as well) on a function, and then solved using this analysis.

User may specify ground truth, which will override all types at certain program points during constraint solving.

__init__(*constraints*, *func_var*, *ground_truth=None*, *var_mapping=None*, *must_struct=None*, *stackvar_max_sizes=None*, *stack_offset_tvs=None*, *constraint_set_degradation_threshold=150*, *type_translator=None*, *tv_manager=None*)

**Parameters**

- **constraints**
- **ground_truth** – A set of SimType-style solutions for some or all type variables. They will be respected during type solving.
- **var_mapping** (dict[*SimVariable*, set[*TypeVariable*]] | None)
- **must_struct** (set[*TypeVariable*] | None)
- **stackvar_max_sizes** (*dict[TypeVariable, int] | None*)
- **stack_offset_tvs** (*dict[int, TypeVariable] | None*)
- **constraint_set_degradation_threshold** (*int*)
- **type_translator** (*TypeTranslator | None*)
- **tv_manager** (*TypeVariableManager | None*)

update_variable_types(*func_addr*, *var_to_typevars*, *stack_offset_tvs=None*)

**Return type**
None

**Parameters**

- **func_addr** (*int | str*)
- **var_to_typevars** (*dict[SimVariable, set[TypeVariable]]*)
- **stack_offset_tvs** (*dict[int, TypeVariable] | None*)

pp_constraints()

Pretty-print constraints between *variables* using the variable mapping.

**Return type**
None

pp_solution()

Pretty-print solutions using the variable mapping.

**Return type**
None

### angr.analyses.typehoon.typevars

class angr.analyses.typehoon.typevars.**TypeConstraint**

Bases: object

pp_str(*mapping*)

**Return type**
str

**Parameters**
**mapping** (*dict[TypeVariable, Any]*)

class angr.analyses.typehoon.typevars.**Equivalence**

Bases: *TypeConstraint*

**__init__**(*type_a*, *type_b*)

**type_a**

**type_b**

**pp_str**(*mapping*)

**Return type**
str

**Parameters**
**mapping** (*dict[TypeVariable, Any]*)

class angr.analyses.typehoon.typevars.**Existence**

Bases: *TypeConstraint*

**__init__**(*type_*)

**type_**

**pp_str**(*mapping*)

**Return type**
str

**Parameters**
**mapping** (*dict[TypeVariable, Any]*)

**replace**(*replacements*)

class angr.analyses.typehoon.typevars.**Subtype**

Bases: *TypeConstraint*

**__init__**(*sub_type*, *super_type*)

**Parameters**

- **sub_type** (TypeConstant | TypeVariable | DerivedTypeVariable)
- **super_type** (TypeConstant | TypeVariable | DerivedTypeVariable)

**super_type**

**sub_type**

**pp_str**(*mapping*)

**Return type**
str

**Parameters**
**mapping** (*dict[TypeVariable, Any]*)

**replace**(*replacements*)

class angr.analyses.typehoon.typevars.**Add**

Bases: *TypeConstraint*

Describes the constraint that type_r == type0 + type1

__init__(*type_0*, *type_1*, *type_r*)

type_0

type_1

type_r

pp_str(*mapping*)

Return type
str

Parameters
mapping (*dict[TypeVariable, Any]*)

replace(*replacements*)

class angr.analyses.typehoon.typevars.Sub

Bases: *TypeConstraint*

Describes the constraint that type_r == type0 - type1

__init__(*type_0*, *type_1*, *type_r*)

type_0

type_1

type_r

pp_str(*mapping*)

Return type
str

Parameters
mapping (*dict[TypeVariable, Any]*)

replace(*replacements*)

class angr.analyses.typehoon.typevars.TypeVariableManager

Bases: object

A manager for creating new type variables and derived type variables and keeping track of their indices.

Usually we create a new TypeVariableManager for each function we analyze.

__init__(*func_addr*, *idx=0*)

Parameters

- func_addr (*int*)
- idx (*int*)

next_id()

Return type
int

property max_tv_id: int

**new_tv**(*name=None*)

Create a new TypeVariable with a managed ID.

**Return type**
*TypeVariable*

**Parameters**
**name** (*str | None*)

**new_dtv**(*type_var*, *, *label=None*, *labels=None*)

Create a new DerivedTypeVariable with a managed ID.

**Return type**
*TypeVariable* | *DerivedTypeVariable* | *TypeConstant*

**Parameters**

- **type_var** (TypeVariable | DerivedTypeVariable | TypeConstant)
- **label** (BaseLabel | *None*)
- **labels** (*Sequence*[BaseLabel] | *None*)

**new_dtv_with_merged_labels**(*type_var*, *, *label=None*, *labels=None*)

Create a new DerivedTypeVariable with the given type variable (or DerivedTypeVariable) and labels. Merge the last AddN and SubN labels when possible.

**Return type**
*TypeVariable* | *DerivedTypeVariable* | *TypeConstant*

**Parameters**

- **type_var** (TypeVariable | TypeConstant)
- **label** (BaseLabel | *None*)
- **labels** (*Sequence*[BaseLabel] | *None*)

**class** angr.analyses.typehoon.typevars.**TypeVariable**

Bases: object

**__init__**(*idx=None*, *name=None*)

**Parameters**

- **idx** (*tuple*[*int*, *int*] | *None*)
- **name** (*str* | *None*)

**idx:** **tuple**[**int**, **int**]

**name**

**pp_str**(*mapping*)

**Return type**
str

**Parameters**
**mapping** (*dict*[TypeVariable, *Any*])

**class** angr.analyses.typehoon.typevars.**DerivedTypeVariable**

Bases: *TypeVariable*

__init__(*type_var*, *label*, *labels=None*, *idx=None*)

**Parameters**

- **type_var** (TypeVariable | DerivedTypeVariable | TypeConstant)
- **label** (BaseLabel | *None*)
- **labels** (*Iterable[BaseLabel]* | *None*)
- **idx** (*tuple[int, int]* | *None*)

**type_var**: *TypeVariable* | *TypeConstant*

**labels**: tuple[*BaseLabel*, ...]

**one_label**()

**Return type**
*BaseLabel* | None

**first_label**()

**Return type**
*BaseLabel*

**path**()

**Return type**
tuple[*BaseLabel*, ...]

**longest_prefix**()

**Return type**
Union[*TypeConstant*, *TypeVariable*, *DerivedTypeVariable*, None]

**pp_str**(*mapping*)

**Return type**
str

**Parameters**
**mapping** (*dict[TypeVariable, Any]*)

**replace**(*replacements*)

class angr.analyses.typehoon.typevars.**TypeVariables**

Bases: object

**__init__**()

**copy**()

**add_type_variable**(*var*, *typevar*, *latest=True*)

**Parameters**

- **var** (SimVariable)
- **typevar** (TypeVariable)
- **latest** (*bool*)

**get_type_variable**(*var*)

**has_type_variable_for**(*var*)

**Parameters**
**var** (SimVariable)

**typevar_to_variable**(*typevar*)

**Return type**
*SimVariable* | None

**Parameters**
**typevar** (TypeVariable)

**class** angr.analyses.typehoon.typevars.**BaseLabel**

Bases: object

**__init__**()

**property variance:** *Variance*

**class** angr.analyses.typehoon.typevars.**FuncIn**

Bases: *BaseLabel*

**__init__**(*loc*)

**loc**

**class** angr.analyses.typehoon.typevars.**FuncOut**

Bases: *BaseLabel*

**__init__**(*loc*)

**loc**

**class** angr.analyses.typehoon.typevars.**Load**

Bases: *BaseLabel*

**class** angr.analyses.typehoon.typevars.**Store**

Bases: *BaseLabel*

**property variance:** *Variance*

**class** angr.analyses.typehoon.typevars.**AddN**

Bases: *BaseLabel*

**__init__**(*n*)

**n**

**class** angr.analyses.typehoon.typevars.**SubN**

Bases: *BaseLabel*

**__init__**(*n*)

**n**

angr.analyses.typehoon.typevars.**add_label**(*value*, *bits*)

Build the label for `ptr + value`, where `value` is a `bits`-wide constant. The constant is interpreted as signed, so that `ptr + 0xffff_ffff_ffff_ffe8` yields `SubN(24)` instead of `AddN(2 ** 64 - 24)`.

**Return type**
*AddN* | *SubN*

**Parameters**

- **value** (*int*)
- **bits** (*int*)

angr.analyses.typehoon.typevars.**sub_label**(*value*, *bits*)

Build the label for `ptr - value`. See *add_label()*.

**Return type**
*AddN* | *SubN*

**Parameters**

- **value** (*int*)
- **bits** (*int*)

**class** angr.analyses.typehoon.typevars.**ConvertTo**

Bases: *BaseLabel*

**__init__**(*to_bits*)

**to_bits**

**class** angr.analyses.typehoon.typevars.**ReinterpretAs**

Bases: *BaseLabel*

**__init__**(*to_type*, *to_bits*)

**to_type**

**to_bits**

**class** angr.analyses.typehoon.typevars.**HasField**

Bases: *BaseLabel*

Has a field of the given bit size * elem_count at the given offset.

**__init__**(*bits*, *offset*, *elem_count=1*)

**Parameters**
**elem_count** (*int*)

**bits**

**offset**

**elem_count**

**class** angr.analyses.typehoon.typevars.**IsArray**

Bases: *BaseLabel*

### angr.analyses.typehoon.variance

**class** angr.analyses.typehoon.variance.**Variance**

Bases: Enum

Enum class describing the variance of type constraints.

__init__(*n*)

COVARIANT = 0

CONTRAVARIANT = 1

### 10.3.50 angr.analyses.unpacker

class angr.analyses.unpacker.ObfuscationDetector

Bases: *Analysis*

This analysis detects, usually in ways that are more robust than section name matching or signature matching, the existence of obfuscation techniques in a binary.

__init__(*cfg=None*)

**Parameters**
**cfg** (*CFGModel | None*)

analyze()

class angr.analyses.unpacker.PackingDetector

Bases: *Analysis*

This analysis detects if a binary is likely packed or not. We may extend it to identify which packer is in use in the future.

PACKED_MIN_BYTES = 256

PACKED_ENTROPY_MIN_THRESHOLD = 0.88

__init__(*cfg=None*, *region_size_threshold=32*)

**Parameters**

- **cfg** (*CFGModel | None*)
- **region_size_threshold** (*int*)

packed: bool

region_size_threshold: int

analyze()

#### Submodules

*obfuscation_detector*
*packing_detector*

#### angr.analyses.unpacker.obfuscation_detector

class angr.analyses.unpacker.obfuscation_detector.ObfuscationDetector

Bases: *Analysis*

This analysis detects, usually in ways that are more robust than section name matching or signature matching, the existence of obfuscation techniques in a binary.

__init__(*cfg=None*)

**Parameters**
**cfg** (CFGModel | *None*)

**analyze**()

### angr.analyses.unpacker.packing_detector

**class** angr.analyses.unpacker.packing_detector.**PackingDetector**

Bases: *Analysis*

This analysis detects if a binary is likely packed or not. We may extend it to identify which packer is in use in the future.

**PACKED_MIN_BYTES = 256**

**PACKED_ENTROPY_MIN_THRESHOLD = 0.88**

__init__(*cfg=None*, *region_size_threshold=32*)

**Parameters**

- **cfg** (CFGModel | *None*)
- **region_size_threshold** (*int*)

**packed:** bool

**region_size_threshold:** int

**analyze**()

## 10.3.51 angr.analyses.variable_recovery

**class** angr.analyses.variable_recovery.**VariableRecovery**

Bases: *ForwardAnalysis*, *VariableRecoveryBase*

Recover "variables" from a function using forced execution.

While variables play a very important role in programming, it does not really exist after compiling. However, we can still identify and recovery their counterparts in binaries. It is worth noting that not every variable in source code can be identified in binaries, and not every recognized variable in binaries have a corresponding variable in the original source code. In short, there is no guarantee that the variables we identified/recognized in a binary are the same variables in its source code.

This analysis uses heuristics to identify and recovers the following types of variables: - Register variables. - Stack variables. - Heap variables. (not implemented yet) - Global variables. (not implemented yet)

This analysis takes a function as input, and performs a data-flow analysis on nodes. It runs concrete execution on every statement and hooks all register/memory accesses to discover all places that are accessing variables. It is slow, but has a more accurate analysis result. For a fast but inaccurate variable recovery, you may consider using VariableRecoveryFast.

This analysis follows SSA, which means every write creates a new variable in registers or memory (statck, heap, etc.). Things may get tricky when overlapping variable (in memory, as you cannot really have overlapping accesses to registers) accesses exist, and in such cases, a new variable will be created, and this new variable will overlap with one or more existing variables. A decision procedure (which is pretty much TODO) is required at the end of this analysis to resolve the conflicts between overlapping variables.

__init__(*func*, *max_iterations=20*, *store_live_variables=False*)

**Parameters**
**func** (*knowledge.Function*) – The function to analyze.

class angr.analyses.variable_recovery.**VariableRecoveryFast**

Bases: *ForwardAnalysis*, *VariableRecoveryBase*

Recover "variables" from a function by keeping track of stack pointer offsets and pattern matching VEX statements.

If calling conventions are recovered prior to running VariableRecoveryFast, variables can be recognized more accurately. However, it is not a requirement. In this case, the function graph you pass must contain information indicating the call-out sites inside the analyzed function. These graph edges must be annotated with either `"type": "call"` or `"outside": True`.

### Submodules


*annotations*
*engine_ail*
*engine_base*
*engine_vex*
*irsb_scanner*
*variable_recovery*
*variable_recovery_base*
*variable_recovery_fast*


### angr.analyses.variable_recovery.annotations

class angr.analyses.variable_recovery.annotations.**StackLocationAnnotation**

Bases: Annotation

**eliminatable = False**

**relocatable = False**

__init__(*offset*)

class angr.analyses.variable_recovery.annotations.**VariableSourceAnnotation**

Bases: Annotation

**eliminatable = False**

**relocatable = False**

__init__(*block_addr*, *stmt_idx*, *ins_addr*)

**static from_state**(*state*)

### angr.analyses.variable_recovery.engine_ail

class angr.analyses.variable_recovery.engine_ail.**SimEngineVRAIL**

Bases: *SimEngineNostmtAIL*[VariableRecoveryFastState, *RichR*[BV | FP], None, None], *SimEngineVRBase*[VariableRecoveryFastState, Block]

The engine for variable recovery on AIL.

__init__(**args*, *type_lifter*, *call_info=None*, *vvar_to_vvar*, *vvar_type_hints=None*, *func_ret_var=None*, *tv_manager=None*, *variable_map=None*, ***kwargs*)

**Parameters**

- **type_lifter** (TypeTranslator)
- **vvar_to_vvar** (*dict[int, int] | None*)
- **vvar_type_hints** (*dict[int, TypeConstant] | None*)
- **func_ret_var** (SimVariable | *None*)
- **tv_manager** (TypeVariableManager | *None*)

### angr.analyses.variable_recovery.engine_base

class angr.analyses.variable_recovery.engine_base.**RichR**

Bases: Generic

A rich representation of calculation results. The variable recovery data domain.

__init__(*data*, *variable=None*, *typevar=None*, *type_constraints=None*)

**Parameters**

- **data** (*RichRT_co*)
- **typevar** (typeconsts.TypeConstant | typevars.TypeVariable | *None*)
- **type_constraints** (*set[typevars.TypeConstraint] | None*)

**data**

**variable**

**typevar:** *TypeConstant* | *TypeVariable* | None

**type_constraints**

**property bits:** int

class angr.analyses.variable_recovery.engine_base.**SimEngineVRBase**

Bases: *SimEngineLight*[VRStateType, *RichR*[BV | FP], BlockType, None], Generic

The base class for variable recovery analyses. Contains methods for basic interactions with the state, like loading and storing data.

__init__(*project*, *kb*, *vvar_type_hints=None*, *tv_manager=None*)

**Parameters**

- **vvar_type_hints** (*dict[int, TypeConstant] | None*)
- **tv_manager** (TypeVariableManager | *None*)

**vvar_type_hints:** dict[int, *TypeConstant*]

**vvar_region:** dict[int, Any]

**property func_addr**

**property block_idx:** int | None

### angr.analyses.variable_recovery.engine_vex

**class** angr.analyses.variable_recovery.engine_vex.**SimEngineVRVEX**

Bases: *SimEngineNostmtVEX*[VariableRecoveryFastState, *RichR*[BV | FP], None], *SimEngineVRBase*[VariableRecoveryFastState, *Block*]

Implements the VEX engine for variable recovery analysis.

**reg_read_stmts_to_ignore:** set[int]

**stmts_to_lower:** set[int]

**__init__**(**args*, *call_info=None*, ***kwargs*)

### angr.analyses.variable_recovery.irsb_scanner

**class** angr.analyses.variable_recovery.irsb_scanner.**VEXIRSBScanner**

Bases: *SimEngineLightVEX*[None, None, None, None]

Scan the VEX IRSB to determine if any argument-passing registers should be narrowed by detecting cases of loading the whole register and immediately narrowing the register before writing to the tmp.

**__init__**(**args*, ***kwargs*)

**process**(*state*, ***, *block=None*, *whitelist=None*, ***kwargs*)

### angr.analyses.variable_recovery.variable_recovery

**class** angr.analyses.variable_recovery.variable_recovery.**VariableRecoveryState**

Bases: *VariableRecoveryStateBase*

The abstract state of variable recovery analysis.

**Variables**
**variable_manager** (*angr.knowledge.variable_manager.VariableManager*) – The variable manager.

**__init__**(*project*, *block_addr*, *analysis*, *arch*, *func*, *concrete_states*, ***, *tv_manager*, *stack_region=None*, *register_region=None*)

**Parameters**

- **project** (Project)
- **block_addr** (*int*)
- **arch** (*Arch*)
- **func** (Function)
- **tv_manager** (TypeVariableManager)

**property concrete_states**

**get_concrete_state**(*addr*)

**Parameters**
**addr**

**Returns**

**copy**()

**register_callbacks**(*concrete_states*)

**Parameters**
**concrete_states**

**Returns**

**merge**(*others*, *successor=None*)

Merge two abstract states.

**Parameters**
**others** (tuple[*VariableRecoveryState*, ...]) – Other abstract states to merge.

**Returns**
The merged abstract state.

**Return type**
tuple[*VariableRecoveryState*, bool]

**class** angr.analyses.variable_recovery.variable_recovery.**VariableRecovery**

Bases: *ForwardAnalysis*, *VariableRecoveryBase*

Recover "variables" from a function using forced execution.

While variables play a very important role in programming, it does not really exist after compiling. However, we can still identify and recovery their counterparts in binaries. It is worth noting that not every variable in source code can be identified in binaries, and not every recognized variable in binaries have a corresponding variable in the original source code. In short, there is no guarantee that the variables we identified/recognized in a binary are the same variables in its source code.

This analysis uses heuristics to identify and recovers the following types of variables: - Register variables. - Stack variables. - Heap variables. (not implemented yet) - Global variables. (not implemented yet)

This analysis takes a function as input, and performs a data-flow analysis on nodes. It runs concrete execution on every statement and hooks all register/memory accesses to discover all places that are accessing variables. It is slow, but has a more accurate analysis result. For a fast but inaccurate variable recovery, you may consider using VariableRecoveryFast.

This analysis follows SSA, which means every write creates a new variable in registers or memory (statck, heap, etc.). Things may get tricky when overlapping variable (in memory, as you cannot really have overlapping accesses to registers) accesses exist, and in such cases, a new variable will be created, and this new variable will overlap with one or more existing variables. A decision procedure (which is pretty much TODO) is required at the end of this analysis to resolve the conflicts between overlapping variables.

**__init__**(*func*, *max_iterations=20*, *store_live_variables=False*)

**Parameters**
**func** (*knowledge.Function*) – The function to analyze.

### angr.analyses.variable_recovery.variable_recovery_base

angr.analyses.variable_recovery.variable_recovery_base.**parse_stack_pointer**(*sp*)

Convert multiple supported forms of stack pointer representations into stack offsets.

**Parameters**
**sp** – A stack pointer representation.

**Returns**
A stack pointer offset.

**Return type**
int

class angr.analyses.variable_recovery.variable_recovery_base.**VariableAnnotation**

Bases: Annotation

**relocatable = True**

**eliminatable = False**

**__init__**(*addr_and_variables*)

**Parameters**
**addr_and_variables** (*list[tuple[int, SimVariable]]*)

**addr_and_variables**

class angr.analyses.variable_recovery.variable_recovery_base.**VariableRecoveryBase**

Bases: *Analysis*

The base class for VariableRecovery and VariableRecoveryFast.

**__init__**(*func*, *max_iterations*, *store_live_variables*, *vvar_to_vvar=None*, *func_graph=None*, *entry_node_addr=None*)

**get_variable_definitions**(*block_addr*)

Get variables that are defined at the specified block.

**Parameters**
**block_addr** (*int*) – Address of the block.

**Returns**
A set of variables.

**initialize_dominance_frontiers**()

class angr.analyses.variable_recovery.variable_recovery_base.**VariableRecoveryStateBase**

Bases: object

The base abstract state for variable recovery analysis.

**__init__**(*block_addr*, *analysis*, *arch*, *func*, *project*, ***, *tv_manager*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=None*, *func_typevar=None*, *delayed_type_constraints=None*, *stack_offset_typevars=None*)

**Parameters**

- **block_addr** (*int*)
- **analysis** (VariableRecoveryBase)
- **arch** (*Arch*)
- **func** (Function)
- **project** (Project)
- **tv_manager** (TypeVariableManager)

**arch:** *Arch*

**stack_region:** *MultiValuedMemory*

**register_region:** *MultiValuedMemory*

**global_region:** *MultiValuedMemory*

successor_block_addr: int | None

phi_variables: dict[SimVariable, SimVariable]

stack_offset_typevars: dict[int, TypeVariable]

static top(*bits*)

**Return type**
BV

static is_top(*thing*)

**Return type**
bool

static extract_variables(*expr*)

**Return type**
Generator[tuple[int, SimVariable]]

**Parameters**
**expr** (*Base*)

static annotate_with_variables(*expr*, *addr_and_variables*)

**Return type**
TypeVar(T, bound= Base)

**Parameters**

- **expr** (*T*)
- **addr_and_variables** (*Iterable[tuple[int, SimVariable]]*)

stack_address(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int*)

static is_stack_address(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

is_global_variable_address(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Bits*)

get_stack_offset(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Bits*)

**stack_addr_from_offset**(*offset*)

**Return type**
int

**Parameters**
**offset** (*int*)

**property func_addr**

**property dominance_frontiers**

**property variable_manager**

**property variables**

**get_variable_definitions**(*block_addr*)
Get variables that are defined at the specified block.

**Parameters**
**block_addr** (*int*) – Address of the block.

**Returns**
A set of variables.

**add_type_constraint**(*constraint*)
Add a new type constraint.

**Parameters**
**constraint**

**Returns**

**add_type_constraint_for_function**(*func_typevar*, *constraint*)
Add a new type constraint for a specified function.

**Parameters**

- **func_typevar**
- **constraint**

**Returns**

**downsize**()
Remove unnecessary members.

**Return type**
None

**Returns**
None

**static downsize_region**(*region*)
Get rid of unnecessary references in region so that it won't avoid garbage collection on those referenced objects.

**Parameters**
**region** (*MultiValuedMemory*) – A MultiValuedMemory region.

**Return type**
*MultiValuedMemory*

**Returns**
None

### angr.analyses.variable_recovery.variable_recovery_fast

**class** angr.analyses.variable_recovery.variable_recovery_fast.**VariableRecoveryFastState**

Bases: *VariableRecoveryStateBase*

The abstract state of variable recovery analysis.

**Variables**

- **stack_region** (KeyedRegion) – The stack store.
- **register_region** (KeyedRegion) – The register store.

**__init__**(*block_addr*, *analysis*, *arch*, *func*, *project*, *stack_region=None*, *register_region=None*, *global_region=None*, *typevars=None*, *type_constraints=None*, *func_typevar=None*, *delayed_type_constraints=None*, *stack_offset_typevars=None*, *ret_val_size=None*, ***, *tv_manager*)

**Parameters**
**tv_manager** (TypeVariableManager)

**copy**()

**merge**(*others*, *successor=None*)

Merge two abstract states.

For any node A whose dominance frontier that the current node (at the current program location) belongs to, we create a phi variable V' for each variable V that is defined in A, and then replace all existence of V with V' in the merged abstract state.

**Parameters**
**others** (tuple[*VariableRecoveryFastState*, ...]) – Other abstract states to merge.

**Return type**
tuple[*VariableRecoveryFastState*, bool]

**Returns**
The merged abstract state.

**class** angr.analyses.variable_recovery.variable_recovery_fast.**VariableRecoveryFast**

Bases: *ForwardAnalysis*, *VariableRecoveryBase*

Recover "variables" from a function by keeping track of stack pointer offsets and pattern matching VEX statements.

If calling conventions are recovered prior to running VariableRecoveryFast, variables can be recognized more accurately. However, it is not a requirement. In this case, the function graph you pass must contain information indicating the call-out sites inside the analyzed function. These graph edges must be annotated with either `"type": "call"` or `"outside": True`.

## 10.3.52 angr.analyses.veritesting

**exception** angr.analyses.veritesting.**VeritestingError**

Bases: Exception

class angr.analyses.veritesting.**CallTracingFilter**

Bases: object

Filter to apply during CFG creation on a given state and jumpkind to determine if it should be skipped at a certain depth

**whitelist = {<class 'angr.procedures.cgc.receive.receive'>, <class 'angr.procedures.cgc.transmit.transmit'>, <class 'angr.procedures.glibc.__ctype_b_loc.__ctype_b_loc'>, <class 'angr.procedures.libc.atoi.atoi'>, <class 'angr.procedures.libc.fgetc.fgetc'>, <class 'angr.procedures.libc.strcmp.strcmp'>, <class 'angr.procedures.libc.strlen.strlen'>, <class 'angr.procedures.posix.read.read'>}**

**cfg_cache = {}**

**__init__**(*project*, *depth*, *blacklist=None*)

**filter**(*call_target_state*, *jumpkind*)

The call will be skipped if it returns True.

**Parameters**

- **call_target_state** – The new state of the call target.
- **jumpkind** – The Jumpkind of this call.

**Returns**

True if we want to skip this call, False otherwise.

class angr.analyses.veritesting.**Veritesting**

Bases: *Analysis*

An exploration technique made for condensing chunks of code to single (nested) if-then-else constraints via CFG accurate to conduct Static Symbolic Execution SSE (conversion to single constraint)

**cfg_cache = {}**

**all_stashes = ('successful', 'errored', 'deadended', 'deviated', 'unconstrained')**

**__init__**(*input_state*, *boundaries=None*, *loop_unrolling_limit=10*, *enable_function_inlining=False*, *terminator=None*, *deviation_filter=None*)

SSE stands for Static Symbolic Execution, and we also implemented an extended version of Veritesting (Avgerinos, Thanassis, et al, ICSE 2014).

**Parameters**

- **input_state** – The initial state to begin the execution with.
- **boundaries** – Addresses where execution should stop.
- **loop_unrolling_limit** – The maximum times that Veritesting should unroll a loop for.
- **enable_function_inlining** – Whether we should enable function inlining and syscall inlining.
- **terminator** – A callback function that takes a state as parameter. Veritesting will terminate if this function returns True.
- **deviation_filter** – A callback function that takes a state as parameter. Veritesting will put the state into "deviated" stash if this function returns True.

`is_not_in_cfg`(*s*)

Returns if s.addr is not a proper node in our CFG.

**Parameters**
**s** (SimState) – The SimState instance to test.

**Returns bool**
False if our CFG contains p.addr, True otherwise.

`is_overbound`(*state*)

Filter out all states that run out of boundaries or loop too many times.

param SimState state: SimState instance to check returns bool: True if outside of mem/loop_ctr boundary

### 10.3.53 angr.analyses.vfg

`class angr.analyses.vfg.VFGJob`

Bases: *CFGJobBase*

A job descriptor that contains local variables used during VFG analysis.

`__init__`(**args*, ***kwargs*)

**Return type**
None

`property block_id: BlockID | None`

`callstack_repr`(*kb*)

**Parameters**
**kb** (KnowledgeBase)

`class angr.analyses.vfg.PendingJob`

Bases: object

Describes a pending job during VFG analysis.

`__init__`(*block_id*, *state*, *call_stack*, *src_block_id*, *src_stmt_idx*, *src_ins_addr*)

**Parameters**

- **block_id** (BlockID)
- **state** (SimState)
- **call_stack** (CallStack)
- **src_block_id** (BlockID)
- **src_stmt_idx** (*int*)
- **src_ins_addr** (*int*)

**Return type**
None

`block_id`

`state`

`call_stack`

**src_block_id**

**src_stmt_idx**

**src_ins_addr**

**class** angr.analyses.vfg.**AnalysisTask**

Bases: object

An analysis task describes a task that should be done before popping this task out of the task stack and discard it.

**__init__**()

**Return type**
None

**property done**

**class** angr.analyses.vfg.**FunctionAnalysis**

Bases: *AnalysisTask*

Analyze a function, generate fix-point states from all endpoints of that function, and then merge them to one state.

**__init__**(*function_address*, *return_address*)

**Parameters**

- **function_address** (*int*)
- **return_address** (*int | None*)

**Return type**
None

**call_analysis:** *AnalysisTask* | None

**property done:** bool

**class** angr.analyses.vfg.**CallAnalysis**

Bases: *AnalysisTask*

Analyze a call by analyze all functions this call might be calling, collect all final states generated by analyzing those functions, and merge them into one state.

**__init__**(*address*, *return_address*, *function_analysis_tasks=None*, *mergeable_plugins=None*)

**Parameters**

- **address** (*int*)
- **return_address** (*None*)
- **function_analysis_tasks** (*list[Any] | None*)
- **mergeable_plugins** (*tuple[str, str] | None*)

**Return type**
None

**property done:** bool

**register_function_analysis**(*task*)

**Return type**
None

**Parameters**
**task** (FunctionAnalysis)

**add_final_job**(*job*)

**Return type**
None

**Parameters**
**job** (VFGJob)

**merge_jobs**()

**Return type**
*VFGJob*

class angr.analyses.vfg.**VFGNode**

Bases: object

A descriptor of nodes in a Value-Flow Graph

**__init__**(*addr*, *key*, *state=None*)

Constructor.

**Parameters**

- **addr** (int)
- **key** (*BlockID*)
- **state** (*SimState* | None)

**Return type**
None

**widened_state:** *SimState* | None

**narrowing_times:** int

**all_states:** list[*SimState*]

**events:** list

**input_variables:** list

**actions:** list

**final_states:** list[*SimState*]

**state:** *SimState* | None

**append_state**(*s*, *is_widened_state=False*)

Appended a new state to this VFGNode. :type s: :param s: The new state to append :type is_widened_state: :param is_widened_state: Whether it is a widened state or not.

**class** angr.analyses.vfg.**VFG**

Bases: *ForwardAnalysis*[*SimState*, *VFGNode*, *VFGJob*, *BlockID*, *SimState*], *Analysis*

This class represents a control-flow graph with static analysis result.

Perform abstract interpretation analysis starting from the given function address. The output is an invariant at the beginning (or the end) of each basic block.

Steps:

- Generate a CFG first if CFG is not provided.
- Identify all merge points (denote the set of merge points as Pw) in the CFG.
- Cut those loop back edges (can be derived from Pw) so that we gain an acyclic CFG.
- **Identify all variables that are 1) from memory loading 2) from initial values, or 3) phi functions. Denote**
  the set of those variables as S_{var}.
- **Start real AI analysis and try to compute a fix point of each merge point. Perform widening/narrowing only on**
  variables in S_{var}.

**__init__**(*cfg=None*, *context_sensitivity_level=2*, *start=None*, *function_start=None*, *interfunction_level=0*, *initial_state=None*, *avoid_runs=None*, *remove_options=None*, *timeout=None*, *max_iterations_before_widening=8*, *max_iterations=40*, *widening_interval=3*, *final_state_callback=None*, *status_callback=None*, *record_function_final_states=False*)

**Parameters**

- **cfg** (*CFGEmulated* | None) – The control-flow graph to base this analysis on. If none is provided, we will construct a CFGEmulated.
- **context_sensitivity_level** (int) – The level of context-sensitivity of this VFG. It ranges from 0 to infinity. Default 2.
- **function_start** (int | None) – The address of the function to analyze.
- **interfunction_level** (int) – The level of interfunction-ness to be
- **initial_state** (*SimState* | None) – A state to use as the initial one
- **avoid_runs** (list[int] | None) – A list of runs to avoid
- **remove_options** (set[str] | None) – State options to remove from the initial state. It only works when *initial_state* is None
- **timeout** (int | None)
- **final_state_callback** (Callable[[*SimState*, *CallStack*], Any] | None) – callback function when countering final state
- **status_callback** (Callable[[*VFG*], Any] | None) – callback function used in _analysis_core_baremetal
- **start** (*int* | *None*)
- **max_iterations_before_widening** (*int*)
- **max_iterations** (*int*)
- **widening_interval** (*int*)
- **record_function_final_states** (*bool*)

**Return type**
None

`final_states: list[SimState]`

`property function_initial_states`

`property function_final_states`

`get_any_node`(*addr*)

Get any VFG node corresponding to the basic block at @addr. Note that depending on the context sensitivity level, there might be multiple nodes corresponding to different contexts. This function will return the first one it encounters, which might not be what you want.

**Return type**
*VFGNode* | None

**Parameters**
**addr** (*int*)

`get_all_nodes`(*addr*)

**Return type**
Generator[*VFGNode*]

`irsb_from_node`(*node*)

`copy`()

### 10.3.54 angr.analyses.vsa_ddg

`class angr.analyses.vsa_ddg.DefUseChain`

Bases: object

Stand for a def-use chain. it is generated by the DDG itself.

`__init__`(*def_loc*, *use_loc*, *variable*)

Constructor.

**Parameters**

- **def_loc**
- **use_loc**
- **variable**

**Returns**

`class angr.analyses.vsa_ddg.VSA_DDG`

Bases: *Analysis*

A Data dependency graph based on VSA states. That means we don't (and shouldn't) expect any symbolic expressions.

`__init__`(*vfg=None*, *start_addr=None*, *interfunction_level=0*, *context_sensitivity_level=2*, *keep_data=False*)

Constructor.

**Parameters**

- **vfg** – An already constructed VFG. If not specified, a new VFG will be created with other specified parameters. *vfg* and *start_addr* cannot both be unspecified.
- **start_addr** – The address where to start the analysis (typically, a function's entry point).
- **interfunction_level** – See VFG analysis.
- **context_sensitivity_level** – See VFG analysis.
- **keep_data** – Whether we keep set of addresses as edges in the graph, or just the cardinality of the sets, which can be used as a "weight".

**get_predecessors**(*code_location*)

Returns all predecessors of *code_location*.

**Parameters**
**code_location** – A CodeLocation instance.

**Returns**
A list of all predecessors.

**get_all_nodes**(*simrun_addr*, *stmt_idx*)

Get all DDG nodes matching the given basic block address and statement index.

### 10.3.55 angr.analyses.vtable

**class** angr.analyses.vtable.**Vtable**

Bases: object

This contains the addr, size and function addresses of a Vtable

**__init__**(*vaddr*, *size*, *func_addrs=None*)

**class** angr.analyses.vtable.**VtableFinder**

Bases: *Analysis*

This analysis locates Vtables in a binary based on heuristics taken from - "Reconstruction of Class Hierarchies for Decompilation of C++ Programs"

**__init__**()

**is_cross_referenced**(*addr*)

**is_function**(*addr*)

**analyze**()

**create_extract_vtable**(*start_addr*, *sec_size*)

### 10.3.56 angr.analyses.xrefs

**class** angr.analyses.xrefs.**SimEngineXRefsVEX**

Bases: *SimEngineNostmtVEX*[None, None, None]

The VEX engine class for XRefs analysis.

**__init__**(*xref_manager*, *project*, *replacements=None*)

**add_xref**(*xref_type*, *from_loc*, *to_loc*)

**static extract_value_if_concrete**(*expr*)

Extract the concrete value from expr if it is a concrete claripy AST.

**Parameters**
**expr** – A claripy AST.

**Return type**
int | None

**Returns**
A concrete value or None if nothing concrete can be extracted.

**class** angr.analyses.xrefs.**XRefsAnalysis**

Bases: *ForwardAnalysis*, *Analysis*

XRefsAnalysis recovers in-depth x-refs (cross-references) in disassembly code.

Here is an example:

```
.text:
000023C8                LDR     R2, =time_now
000023CA                LDR     R3, [R2]
000023CC                ADDS    R3, #1
000023CE                STR     R3, [R2]
000023D0                BX      LR

.bss:
1FFF36F4 time_now       % 4
```

You will have the following x-refs for time_now:

```
23c8 - offset
23ca - read access
23ce - write access
```

## 10.4 angr.angrdb

**class** angr.angrdb.**AngrDB**

Bases: object

AngrDB provides a storage solution for an angr project, its knowledge bases, and some other types of data. It is designed to use an SQL-based database as the storage backend.

**ALL_TABLES = ['objects']**

**VERSION = 1**

**__init__**(*project=None*, *nullpool=False*)

**open_db**(*db_str='sqlite:///:memory:'*)

**static session_scope**(*Session*)

**static save_info**(*session*, *key*, *value*)

Save an information entry to the database.

**Parameters**

- **session**
- **key**
- **value**

**Returns**

static loaded_type_collection_names()

Primary names of all loaded type collections.

**Return type**
list[str]

static load_type_collections(*names*)

Make sure the type collections *names* are loaded; warn about the ones that cannot be found.

**Return type**
None

**Parameters**
**names** (*list[str]*)

static get_info(*session*, *key*)

Get an information entry from the database.

**Parameters**

- **session**
- **key**

**Returns**

update_dbinfo(*session*, *extra_info=None*)

Update the information in database.

**Parameters**

- **session**
- **extra_info** (*dict[str, str] | None*)

**Returns**

get_dbinfo(*session*, *extra_info=None*)

Get database information.

**Parameters**

- **session**
- **extra_info** (*dict[str, str] | None*)

**Returns**
A dict of information entries.

db_compatible(*version*)

Checks if the given database version is compatible with the current AngrDB class.

**Parameters**
**version** (*int*) – The version of the database.

**Returns**
True if compatible, False otherwise.

**Return type**
bool

**invalidate**()

**dump**(*db_path*, *kbs=None*, *extra_info=None*)

**Parameters**

- **kbs** (*list[KnowledgeBase] | None*)
- **extra_info** (*dict[str, Any] | None*)

**load**(*db_path*, *kb_names=None*, *other_kbs=None*, *extra_info=None*)

**Parameters**

- **db_path** (*str*)
- **kb_names** (*list[str] | None*)
- **other_kbs** (*dict[str, KnowledgeBase] | None*)
- **extra_info** (*dict[str, Any] | None*)

### Submodules


*db*
*models*
*serializers*


## 10.4.1 angr.angrdb.db

**class** angr.angrdb.db.**AngrDB**

Bases: object

AngrDB provides a storage solution for an angr project, its knowledge bases, and some other types of data. It is designed to use an SQL-based database as the storage backend.

**ALL_TABLES = ['objects']**

**VERSION = 1**

**__init__**(*project=None*, *nullpool=False*)

**open_db**(*db_str='sqlite:///:memory:'*)

**static session_scope**(*Session*)

**static save_info**(*session*, *key*, *value*)

Save an information entry to the database.

**Parameters**

- **session**
- **key**
- **value**

**Returns**

**static loaded_type_collection_names**()

Primary names of all loaded type collections.

**Return type**
list[str]

**static load_type_collections**(*names*)

Make sure the type collections *names* are loaded; warn about the ones that cannot be found.

**Return type**
None

**Parameters**
**names** (*list[str]*)

**static get_info**(*session*, *key*)

Get an information entry from the database.

**Parameters**

- **session**
- **key**

**Returns**

**update_dbinfo**(*session*, *extra_info=None*)

Update the information in database.

**Parameters**

- **session**
- **extra_info** (*dict[str, str] | None*)

**Returns**

**get_dbinfo**(*session*, *extra_info=None*)

Get database information.

**Parameters**

- **session**
- **extra_info** (*dict[str, str] | None*)

**Returns**
A dict of information entries.

**db_compatible**(*version*)

Checks if the given database version is compatible with the current AngrDB class.

**Parameters**
**version** (*int*) – The version of the database.

**Returns**
True if compatible, False otherwise.

**Return type**
bool

**invalidate**()

**dump**(*db_path*, *kbs=None*, *extra_info=None*)

**Parameters**

- **kbs** (*list[KnowledgeBase] | None*)
- **extra_info** (*dict[str, Any] | None*)

**load**(*db_path*, *kb_names=None*, *other_kbs=None*, *extra_info=None*)

**Parameters**

- **db_path** (*str*)
- **kb_names** (*list[str] | None*)
- **other_kbs** (*dict[str, KnowledgeBase] | None*)
- **extra_info** (*dict[str, Any] | None*)

### 10.4.2 angr.angrdb.models

**class** angr.angrdb.models.**DbInformation**

Bases: Base

Stores information related to the current database. Basically a key-value store.

**id**

**key**

**value**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

**class** angr.angrdb.models.**DbObject**

Bases: Base

Models a binary object.

**id**

**main_object**

**path**

**content**

**backend**

**backend_args**

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.DbKnowledgeBase

Bases: Base

Models a knowledge base.

id

name

cfgs

funcs

callgraphs

xrefs

comments

bookmarks

labels

var_collections

dec_var_collections

structured_code

decompilation_caches

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.DbCFGModel

Bases: Base

Models a CFGFast instance.

id

kb_id

kb

ident

blob

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.DbFunction

Bases: Base

Models a Function instance.

id

kb_id

kb

addr

blob

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.DbCallGraph

Bases: Base

Models the callgraph of a function manager.

id

kb_id

kb

blob

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.DbVariableCollection

Bases: Base

Models a VariableManagerInternal instance.

id

kb_id

kb

**func_addr**

**ident**

**blob**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

**class** angr.angrdb.models.**DbDecVariableCollection**

Bases: `Base`

Models a VariableManagerInternal instance of the decompilation variable manager (kb.dec_variables). A separate table from `variables` (the disassembly-level kb.variables) so databases created before it existed remain loadable: `create_all` adds missing tables.

**id**

**kb_id**

**kb**

**func_addr**

**ident**

**blob**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

**class** angr.angrdb.models.**DbStructuredCode**

Bases: `Base`

Models a StructuredCode instance.

**id**

**kb_id**

**kb**

**func_addr**

**flavor**

**expr_comments**

**stmt_comments**

**configuration**

**const_formats**

**ite_exprs**

**errors**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

**class** angr.angrdb.models.**DbDecompilationCache**

Bases: `Base`

Models a fully serialized DecompilationCache instance (a protobuf DecompilationCache message). A separate table from `structured_code` (which stores only codegen metadata) so databases created before it existed remain loadable: `create_all` adds missing tables.

**id**

**kb_id**

**kb**

**func_addr**

**flavor**

**blob**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

**class** angr.angrdb.models.**DbXRefs**

Bases: `Base`

Models an XRefManager instance.

**id**

**kb_id**

**kb**

**blob**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.**DbComment**

Bases: Base

Models a comment.

**id**

**kb_id**

**kb**

**addr**

**comment**

**type**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.**DbBookmark**

Bases: Base

Models a bookmark.

**id**

**kb_id**

**kb**

**addr**

**label**

**created_at**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

class angr.angrdb.models.**DbLabel**

Bases: Base

Models a label.

**id**

**kb_id**

**kb**

**addr**

**name**

**__init__**(***kwargs*)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in `kwargs`.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

### 10.4.3 angr.angrdb.serializers

**class** angr.angrdb.serializers.**KnowledgeBaseSerializer**

Bases: object

Serialize/unserialize a KnowledgeBase object.

**static dump**(*session*, *kb*)

**Parameters**

- **session** – The database session object.
- **kb** (*KnowledgeBase*) – The KnowledgeBase instance to serialize.

**Returns**
None

**static load**(*session*, *project*, *name*)

**Parameters**
**session**

**Returns**

**class** angr.angrdb.serializers.**LoaderSerializer**

Bases: object

Serialize/unserialize a CLE Loader object into/from an angr DB.

Corner cases:

- For certain backends (e.g., CART), we do not store the data of the main object. angr will unpack the CART file again after loading the database.

**NO_MAINBIN_BACKENDS = [<class 'cle.backends.cartfile.CARTFile'>]**

**LOAD_ARG_BLACKLIST = {'is_main_bin', 'loader'}**

backend2name = {<class 'cle.backends.blob.Blob'>: 'blob', <class 'cle.backends.cartfile.CARTFile'>: 'cart', <class 'cle.backends.cgc.backedcgc.BackedCGC'>: 'backedcgc', <class 'cle.backends.cgc.cgc.CGC'>: 'cgc', <class 'cle.backends.coff.Coff'>: 'COFF', <class 'cle.backends.elf.elf.ELF'>: 'elf', <class 'cle.backends.elf.elfcore.ELFCore'>: 'elfcore', <class 'cle.backends.ihex.Hex'>: 'hex', <class 'cle.backends.java.apk.Apk'>: 'apk', <class 'cle.backends.java.dex.Dex'>: 'dex', <class 'cle.backends.java.jar.Jar'>: 'jar', <class 'cle.backends.macho.macho.MachO'>: 'mach-o', <class 'cle.backends.minidump.Minidump'>: 'minidump', <class 'cle.backends.mz.MZ'>: 'mz', <class 'cle.backends.named_region.NamedRegion'>: 'named_region', <class 'cle.backends.pe.pe.PE'>: 'pe', <class 'cle.backends.srec.SRec'>: 'srec', <class 'cle.backends.static_archive.StaticArchive'>: 'AR', <class 'cle.backends.stm32.STM32Backend'>: 'stm32', <class 'cle.backends.te.TE'>: 'te', <class 'cle.backends.uefi_firmware.UefiFirmware'>: 'uefi', <class 'cle.backends.universal2.Universal2'>: 'Universal2', <class 'cle.backends.xbe.XBE'>: 'xbe'}

static json_serialize_load_args(*load_args*)

**Return type**
str

**Parameters**
**load_args** (*dict[str, Any]*)

static should_skip_main_binary(*loader*)

**Return type**
tuple[bool, Backend | None]

static dump(*session*, *loader*)

static load(*session*)

### Submodules


*bookmarks*
*callgraph*
*cfg_model*
*comments*
*funcs*
*kb*
*labels*
*loader*
*structured_code*
*variables*
*xrefs*


### angr.angrdb.serializers.bookmarks

class angr.angrdb.serializers.bookmarks.**BookmarksSerializer**

Bases: object

Serialize/unserialize bookmarks to/from a database session.

static **dump**(*session*, *db_kb*, *bookmarks*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **bookmarks** (Bookmarks)

**Returns**
None

static **load**(*session*, *db_kb*, *kb*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **kb** (KnowledgeBase)

**Returns**

### angr.angrdb.serializers.callgraph

class angr.angrdb.serializers.callgraph.**CallGraphSerializer**

Bases: object

Serialize/unserialize the callgraph of a function manager (a networkx.MultiDiGraph whose nodes are function addresses).

The graph is stored as a JSON document with the exact node set as well as all edges, including their keys and data dicts, so that edge multiplicity and edge data round-trip exactly.

static **dump**(*session*, *db_kb*, *callgraph*)

**Parameters**

- **session**
- **db_kb** (*DbKnowledgeBase*) – The database object for KnowledgeBase.
- **callgraph** (MultiDiGraph) – The callgraph to dump.

**Return type**
None

static **load**(*session*, *db_kb*)

**Parameters**

- **session**
- **db_kb** (*DbKnowledgeBase*) – The database object for KnowledgeBase.

**Return type**
MultiDiGraph | None

**Returns**
The loaded callgraph, or None if no callgraph is stored in the database (e.g., for databases created before callgraph serialization was introduced).

### angr.angrdb.serializers.cfg_model

**class** angr.angrdb.serializers.cfg_model.**CFGModelSerializer**

Bases: object

Serialize/unserialize a CFGModel.

**static dump**(*session*, *db_kb*, *ident*, *cfg_model*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase) – The database object for KnowledgeBase.
- **ident** (*str*) – Identifier of the CFG model.
- **cfg_model** (CFGModel) – The CFG model to dump.

**Returns**
None

**static load**(*session*, *db_kb*, *ident*, *cfg_manager*, *loader=None*)

### angr.angrdb.serializers.comments

**class** angr.angrdb.serializers.comments.**CommentsSerializer**

Bases: object

Serialize/unserialize comments to/from a database session.

**static dump**(*session*, *db_kb*, *comments*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **comments** (Comments)

**Returns**
None

**static load**(*session*, *db_kb*, *kb*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **kb** (KnowledgeBase)

**Returns**

### angr.angrdb.serializers.funcs

**class** angr.angrdb.serializers.funcs.**FunctionManagerSerializer**

Bases: object

Serialize/unserialize a function manager and its functions.

static **dump**(*session*, *db_kb*, *func_manager*)

**Parameters**

- **session**
- **db_kb** (*DbKnowledgeBase*)
- **func_manager** (*FunctionManager*)

**Returns**

static **load**(*session*, *db_kb*, *kb*, *callgraph=None*, *cfg_model=None*)

**Parameters**

- **session**
- **db_kb** (*DbKnowledgeBase*)
- **kb** (*KnowledgeBase*)
- **callgraph** (MultiDiGraph | None) – A deserialized callgraph (if the database stores one). When provided, it is used directly and the (much slower) callgraph rebuilding logic is skipped.
- **cfg_model** (*CFGModel* | None) – An optional CFG model. When provided, the function_address member of all CFG nodes that belong to a function will be set accordingly.

**Returns**
A loaded function manager.

### angr.angrdb.serializers.kb

class angr.angrdb.serializers.kb.**KnowledgeBaseSerializer**

Bases: object

Serialize/unserialize a KnowledgeBase object.

static **dump**(*session*, *kb*)

**Parameters**

- **session** – The database session object.
- **kb** (*KnowledgeBase*) – The KnowledgeBase instance to serialize.

**Returns**
None

static **load**(*session*, *project*, *name*)

**Parameters**
**session**

**Returns**

### angr.angrdb.serializers.labels

class angr.angrdb.serializers.labels.**LabelsSerializer**

Bases: object

Serialize/unserialize labels to/from a database session.

**static dump**(*session*, *db_kb*, *labels*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **labels** (Labels)

**Returns**
None

**static load**(*session*, *db_kb*, *kb*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **kb** (KnowledgeBase)

**Returns**

### angr.angrdb.serializers.loader

**class** angr.angrdb.serializers.loader.**LoadArgsJSONEncoder**

Bases: JSONEncoder

A JSON encoder that supports serializing bytes.

**class** angr.angrdb.serializers.loader.**LoadArgsJSONDecoder**

Bases: JSONDecoder

A JSON decoder that supports unserializing into bytes.

**class** angr.angrdb.serializers.loader.**LoaderSerializer**

Bases: object

Serialize/unserialize a CLE Loader object into/from an angr DB.

Corner cases:

- For certain backends (e.g., CART), we do not store the data of the main object. angr will unpack the CART file again after loading the database.

**NO_MAINBIN_BACKENDS = [<class 'cle.backends.cartfile.CARTFile'>]**

**LOAD_ARG_BLACKLIST = {'is_main_bin', 'loader'}**

backend2name = {<class 'cle.backends.blob.Blob'>: 'blob', <class 'cle.backends.cartfile.CARTFile'>: 'cart', <class 'cle.backends.cgc.backedcgc.BackedCGC'>: 'backedcgc', <class 'cle.backends.cgc.cgc.CGC'>: 'cgc', <class 'cle.backends.coff.Coff'>: 'COFF', <class 'cle.backends.elf.elf.ELF'>: 'elf', <class 'cle.backends.elf.elfcore.ELFCore'>: 'elfcore', <class 'cle.backends.ihex.Hex'>: 'hex', <class 'cle.backends.java.apk.Apk'>: 'apk', <class 'cle.backends.java.dex.Dex'>: 'dex', <class 'cle.backends.java.jar.Jar'>: 'jar', <class 'cle.backends.macho.macho.MachO'>: 'mach-o', <class 'cle.backends.minidump.Minidump'>: 'minidump', <class 'cle.backends.mz.MZ'>: 'mz', <class 'cle.backends.named_region.NamedRegion'>: 'named_region', <class 'cle.backends.pe.pe.PE'>: 'pe', <class 'cle.backends.srec.SRec'>: 'srec', <class 'cle.backends.static_archive.StaticArchive'>: 'AR', <class 'cle.backends.stm32.STM32Backend'>: 'stm32', <class 'cle.backends.te.TE'>: 'te', <class 'cle.backends.uefi_firmware.UefiFirmware'>: 'uefi', <class 'cle.backends.universal2.Universal2'>: 'Universal2', <class 'cle.backends.xbe.XBE'>: 'xbe'}

static json_serialize_load_args(*load_args*)

**Return type**
str

**Parameters**
**load_args** (*dict[str, Any]*)

static should_skip_main_binary(*loader*)

**Return type**
tuple[bool, Backend | None]

static dump(*session*, *loader*)

static load(*session*)

### angr.angrdb.serializers.structured_code

class angr.angrdb.serializers.structured_code.**ConstFormatsSerializer**

Bases: object

Serialize/deserialize the constant formats dictionary.

static to_json(*const_formats*)

**Return type**
dict[int, dict[str, dict[str, bool]]]

**Parameters**
**const_formats** (*dict[tuple[int, int, str], dict[str, bool]]*)

static from_json(*data*)

**Return type**
dict[tuple[int, int, str], dict[str, bool]]

**Parameters**
**data** (*dict[int, dict[str, dict[str, str | bool]]]*)

class angr.angrdb.serializers.structured_code.**StructuredCodeManagerSerializer**

Bases: object

Serialize/deserialize a structured code manager.

**static dump**(*session*, *db_kb*, *code_manager*)

Store every decompilation cache as its fully serialized protobuf bytes (the `decompilation_caches` table). Caches that cannot be serialized (e.g. dummy or rust-flavor codegens) fall back to the legacy `structured_code` rows storing codegen metadata only.

**Parameters**

- **session**
- **db_kb** (*DbKnowledgeBase*)
- **code_manager** (*StructuredCodeManager*)

**Returns**

**static dict_strkey_to_intkey**(*d*)

**Return type**
dict[int, Any]

**Parameters**
**d** (*dict[str, Any]*)

**static load**(*session*, *db_kb*, *kb*)

Load decompilation caches: fully serialized caches from the `decompilation_caches` table first, then legacy `structured_code` rows (from this database or from old databases) for any key not already loaded.

**Parameters**

- **session**
- **db_kb** (*DbKnowledgeBase*)
- **kb** (*KnowledgeBase*)

**Return type**
*StructuredCodeManager*

**Returns**
A loaded structured code manager

### angr.angrdb.serializers.variables

class angr.angrdb.serializers.variables.**VariableManagerSerializer**

Bases: object

Serialize/unserialize a variable manager and its variables.

The same machinery serializes the disassembly-level manager (`kb.variables` into the `variables` table) and the decompilation manager (`kb.dec_variables` into the `dec_variables` table); the target table and manager class are parameterized.

**static dump**(*session*, *db_kb*, *var_manager*, *table=<class 'angr.angrdb.models.DbVariableCollection'>*)

**Parameters**

- **db_kb** (DbKnowledgeBase)
- **var_manager** (VariableManager)

static **dump_dvars**(*session*, *db_kb*, *var_manager*)

**Parameters**

- **db_kb** (DbKnowledgeBase)
- **var_manager** (VariableManager)

static **load**(*session*, *db_kb*, *kb*, *ident=None*, *table=<class 'angr.angrdb.models.DbVariableCollection'>*)

**Parameters**

- **db_kb** (DbKnowledgeBase)
- **kb** (KnowledgeBase)

static **load_dvars**(*session*, *db_kb*, *kb*, *ident=None*)

**Parameters**

- **db_kb** (DbKnowledgeBase)
- **kb** (KnowledgeBase)

static **load_internal**(*db_varcoll*, *variable_manager*)

**Return type**
*VariableManagerInternal*

**Parameters**
**variable_manager** (VariableManager)

### angr.angrdb.serializers.xrefs

class angr.angrdb.serializers.xrefs.**XRefsSerializer**

Bases: object

Serialize/unserialize an XRefs object to/from a database session.

static **dump**(*session*, *db_kb*, *xrefs*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **xrefs** (XRefManager)

**Returns**

static **load**(*session*, *db_kb*, *kb*, *cfg_model=None*)

**Parameters**

- **session**
- **db_kb** (DbKnowledgeBase)
- **kb** (KnowledgeBase)
- **cfg_model** (CFGModel)

**Returns**

## 10.5 angr.annocfg

**class** angr.annocfg.**AnnotatedCFG**

Bases: object

AnnotatedCFG is a control flow graph with statement whitelists and exit whitelists to describe a slice of the program.

**__init__**(*project*, *cfg=None*, *detect_loops=False*)

Constructor.

**Parameters**

- **project** – The angr Project instance
- **cfg** – Control flow graph.
- **detect_loops**

**from_digraph**(*digraph*)

Initialize this AnnotatedCFG object with a networkx.DiGraph consisting of the following form of nodes:

Tuples like (block address, statement ID)

Those nodes are connected by edges indicating the execution flow.

**Parameters**
**digraph** (*networkx.DiGraph*) – A networkx.DiGraph object

**get_addr**(*run*)

**add_block_to_whitelist**(*block*)

**add_statements_to_whitelist**(*block*, *stmt_ids*)

**add_exit_to_whitelist**(*run_from*, *run_to*)

**set_last_statement**(*block_addr*, *stmt_id*)

**add_loop**(*loop_tuple*)

A loop tuple contains a series of IRSB addresses that form a loop. Ideally it always starts with the first IRSB that we meet during the execution.

**should_take_exit**(*addr_from*, *addr_to*)

**should_execute_statement**(*addr*, *stmt_id*)

**get_run**(*addr*)

**get_whitelisted_statements**(*addr*)

**Return type**
list[int] | None

**Returns**
True if all statements are whitelisted

**get_last_statement_index**(*addr*)

Get the statement index of the last statement to execute in the basic block specified by *addr*.

**Parameters**
**addr** (*int*) – Address of the basic block.

**Returns**
The statement index of the last statement to be executed in the block. Usually if the default exit is taken, it will be the last statement to execute. If the block is not in the slice or we should never take any exit going to this block, None is returned.

**Return type**
int or None

**get_loops**()

**get_targets**(*source_addr*)

**dbg_repr**()

**dbg_print_irsb**(*irsb_addr*, *project=None*)
Pretty-print an IRSB with whitelist information

**keep_path**(*path*)
Given a path, returns True if the path should be kept, False if it should be cut.

**merge_points**(*path*)

**successor_func**(*path*)
Callback routine that takes in a path, and returns all feasible successors to path group. This callback routine should be passed to the keyword argument "successor_func" of PathGroup.step().

**Parameters**
**path** – A Path instance.

**Returns**
A list of all feasible Path successors.

## 10.6 angr.blade

**exception** angr.blade.**BadJumpkindNotification**
Bases: Exception

Notifies the caller that the jumpkind is bad (e.g., Ijk_NoDecode)

**class** angr.blade.**Blade**
Bases: object

Blade is a light-weight program slicer that works with networkx DiGraph containing CFGNodes. It is meant to be used in angr for small or on-the-fly analyses.

**__init__**(*graph*, *dst_run*, *dst_stmt_idx*, *direction='backward'*, *project=None*, *cfg=None*, *ignore_sp=False*, *ignore_bp=False*, *ignored_regs=None*, *max_level=3*, *base_state=None*, *stop_at_calls=False*, *cross_insn_opt=False*, *max_predecessors=10*, *include_imarks=True*, *control_dependence=True*)

**Parameters**

- **graph** (DiGraph) – A graph representing the control flow graph. Note that it does not take angr.analyses.CFGEmulated or angr.analyses.CFGFast.
- **dst_run** (int) – An address specifying the target SimRun.
- **dst_stmt_idx** (int) – The target statement index. -1 means executing until the last statement.
- **direction** (str) – 'backward' or 'forward' slicing. Forward slicing is not yet supported.

- **project** (*angr.Project*) – The project instance.
- **cfg** (*angr.analyses.CFGBase*) – the CFG instance. It will be made mandatory later.
- **ignore_sp** (bool) – Whether the stack pointer should be ignored in dependency tracking. Any dependency from/to stack pointers will be ignored if this options is True.
- **ignore_bp** (bool) – Whether the base pointer should be ignored or not.
- **max_level** (int) – The maximum number of blocks that we trace back for.
- **stop_at_calls** (bool) – Limit slicing within a single function. Do not proceed when encounters a call edge.
- **include_imarks** (bool) – Should IMarks (instruction boundaries) be included in the slice.
- **control_dependence** (bool) – Whether to consider control dependencies. If True, the temps controlling conditional exits will be added to the tainting set.
- **max_predecessors** (*int*)

**Returns**
None

**property slice**

**dbg_repr**(*arch=None*)

## 10.7 angr.block

**class** angr.block.**DisassemblerBlock**

Bases: object

Helper class to represent a block of disassembled target architecture instructions

**__init__**(*addr*, *insns*, *thumb*, *arch*)

**addr**

**insns**

**thumb**

**arch**

**pp**()

**class** angr.block.**DisassemblerInsn**

Bases: object

Helper class to represent a disassembled target architecture instruction

**property size: int**

**property address: int**

**property mnemonic: str**

**property op_str: str**

class angr.block.**CapstoneBlock**

Bases: *DisassemblerBlock*

Deep copy of the capstone blocks, which have serious issues with having extended lifespans outside of capstone itself

class angr.block.**CapstoneInsn**

Bases: *DisassemblerInsn*

Represents a capstone instruction.

**__init__**(*capstone_insn*)

**insn**

**property size:** int

**property address:** int

**property mnemonic:** str

**property op_str:** str

class angr.block.**PCodeBlock**

Bases: *DisassemblerBlock*

Deep copy of the pcode blocks

class angr.block.**PCodeInsn**

Bases: *DisassemblerInsn*

Represents a capstone instruction.

**__init__**(*pcode_insn*)

**insn**

**property size:** int

**property address:** int

**property mnemonic:** str

**property op_str:** str

class angr.block.**Block**

Bases: *Serializable*

Represents a basic block in a binary or a program.

**BLOCK_MAX_SIZE = 4096**

**__init__**(*addr*, *project=None*, *arch=None*, *size=None*, *max_size=None*, *byte_string=None*, *thumb=False*, *backup_state=None*, *extra_stop_points=None*, *opt_level=None*, *num_inst=None*, *traceflags=0*, *strict_block_end=None*, *collect_data_refs=False*, *cross_insn_opt=True*, *load_from_ro_regions=False*, *const_prop=False*, *initial_regs=None*, *skip_stmts=False*)

**Parameters**
**arch** (*Arch | None*)

**arch**

**addr**

**thumb**

**size**

**pp**(***kwargs*)

**set_initial_regs**()

**static reset_initial_regs**()

**property vex:** *IRSB* | *IRSB*

**property vex_nostmt**

**property disassembly:** *DisassemblerBlock*

Provide a disassembly object using whatever disassembler is available

**property capstone:** *CapstoneBlock*

**property pcode:** *PCodeBlock*

**property codenode**

**property bytes:** *bytes* | None

**property instructions:** int

**property instruction_addrs**

**class** angr.block.**SootBlock**

Bases: object

Represents a Soot IR basic block.

**__init__**(*addr*, ***, *project*, *arch*)

**Parameters**

- **project** (Project)
- **arch** (*Arch*)

**property soot**

**property size**

**property codenode**

## 10.8 angr.callable

**class** angr.callable.**Callable**

Bases: object

Callable is a representation of a function in the binary that can be interacted with like a native python function.

If you set perform_merge=True (the default), the result will be returned to you, and you can get the result state with callable.result_state.

Otherwise, you can get the resulting simulation manager at callable.result_path_group.

**__init__**(*project*, *addr*, *prototype=None*, *concrete_only=False*, *perform_merge=True*, *base_state=None*, *toc=None*, *cc=None*, *add_options=None*, *remove_options=None*, *step_limit=None*, *techniques=None*)

**Parameters**

- **project** (*Project*) – The project to operate on
- **addr** (int) – The address of the function to use
- **prototype** (*SimTypeFunction* | None) – The signature of the calls you would like to make. This really shouldn't be optional.
- **concrete_only** – Throw an exception if the execution splits into multiple paths
- **perform_merge** – Merge all result states into one at the end (only relevant if concrete_only=False)
- **base_state** (*SimState* | None) – The state from which to do these runs
- **toc** – The address of the table of contents for ppc64
- **cc** (*SimCC* | None) – The SimCC to use for a calling convention
- **add_options** (*set[str] | None*)
- **remove_options** (*set[str] | None*)
- **step_limit** (*int | None*)
- **techniques** (*list[ExplorationTechnique] | None*)

**set_base_state**(*state*)

Swap out the state you'd like to use to perform the call.

**Parameters**
**state** – The state to use to perform the call

**perform_call**(**args*, *prototype=None*)

**call_c**(*c_args*)

Call this Callable with a string of C-style arguments.

**Parameters**
**c_args** (str) – C-style arguments.

**Returns**
The return value from the call.

**Return type**
claripy.Ast

## 10.9 angr.calling_conventions

**class** angr.calling_conventions.**PointerWrapper**

Bases: object

**__init__**(*value*, *buffer=False*)

**class** angr.calling_conventions.**AllocHelper**

Bases: object

**__init__**(*ptrsize*)

**alloc**(*size*)

**dump**(*val*, *state*, *loc=None*)

**translate**(*val*, *base*)

**apply**(*state*, *base*)

**size**()

**classmethod calc_size**(*val*, *arch*)

**classmethod stack_loc**(*val*, *arch*, *offset=0*)

angr.calling_conventions.**refine_locs_with_struct_type**(*arch*, *locs*, *arg_type*, *offset=0*, *treat_bot_as_int=True*, *treat_unsupported_as_int=True*)

**Parameters**

- **arch** (*Arch*)
- **locs** (*list*)
- **arg_type** (SimType)
- **offset** (*int*)

**class** angr.calling_conventions.**SerializableIterator**

Bases: object

**getstate**()

**setstate**(*state*)

**class** angr.calling_conventions.**SerializableListIterator**

Bases: *SerializableIterator*

**__init__**(*lst*)

**getstate**()

**setstate**(*state*)

**class** angr.calling_conventions.**SerializableCounter**

Bases: *SerializableIterator*

**__init__**(*start*, *stride*, *mapping=<function SerializableCounter.<lambda>>*)

**getstate**()

**setstate**(*state*)

**class** angr.calling_conventions.**SimFunctionArgument**

Bases: object

Represent a generic function argument.

**Variables**

- **size** (*int*) – The size of the argument, in number of bytes.

- **is_fp** (*bool*) – Whether loads from this location should return a floating point bitvector

**__init__**(*size*, *is_fp=False*)

**Parameters**

- **size** (*int*)
- **is_fp** (*bool*)

**check_value_set**(*value*, *arch*)

**check_value_get**(*value*)

**set_value**(*state*, *value*, ***kwargs*)

**get_value**(*state*, ***kwargs*)

**refine**(*size*, *arch=None*, *offset=None*, *is_fp=None*)

**get_footprint**()

Return a list of SimRegArg and SimStackArgs that are the base components used for this location

**Return type**
Iterable[*SimRegArg* | *SimStackArg*]

**class** angr.calling_conventions.**SimRegArg**

Bases: *SimFunctionArgument*

Represents a function argument that has been passed in a register.

**Variables**

- **reg_name** (*string*) – The name of the represented register.
- **size** (*int*) – The size of the data to store, in number of bytes.
- **reg_offset** – The offset into the register to start storing data.
- **clear_entire_reg** – Whether a store to this register should zero the unused parts of the register.
- **is_fp** (*bool*) – Whether loads from this location should return a floating point bitvector

**__init__**(*reg_name*, *size*, *reg_offset=0*, *is_fp=False*, *clear_entire_reg=False*)

**Parameters**

- **reg_name** (*str*)
- **size** (*int*)

**check_offset**(*arch*)

**Return type**
int

**set_value**(*state*, *value*, ***kwargs*)

**get_value**(*state*, ***kwargs*)

**refine**(*size*, *arch=None*, *offset=None*, *is_fp=None*)

**sse_extend**()

class angr.calling_conventions.**SimStackArg**

Bases: *SimFunctionArgument*

Represents a function argument that has been passed on the stack.

**Variables**

- **stack_offset** (*int*) – The position of the argument relative to the stack pointer after the function prelude.
- **size** (*int*) – The size of the argument, in number of bytes.
- **is_fp** (*bool*) – Whether loads from this location should return a floating point bitvector

**__init__**(*stack_offset*, *size*, *is_fp=False*)

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **is_fp** (*bool*)

**stack_offset:** int

**set_value**(*state*, *value*, *stack_base=None*, ***kwargs*)

**get_value**(*state*, *stack_base=None*, ***kwargs*)

**refine**(*size*, *arch=None*, *offset=None*, *is_fp=None*)

class angr.calling_conventions.**SimComboArg**

Bases: *SimFunctionArgument*, Generic

An argument which spans multiple storage locations. Locations should be given least-significant first.

**__init__**(*locations*, *is_fp=False*)

**Parameters**
**locations** (*list[T]*)

**set_value**(*state*, *value*, ***kwargs*)

**get_value**(*state*, ***kwargs*)

class angr.calling_conventions.**SimStructArg**

Bases: *SimFunctionArgument*

An argument which de/serializes a struct from a list of storage locations

**Variables**

- **struct** – The simtype describing the structure
- **locs** – The storage locations to use

**__init__**(*struct*, *locs*)

**Parameters**

- **struct** (SimStruct)
- **locs** (*dict[str, SimFunctionArgument]*)

**get_single_footprint**()

**Return type**
*SimStackArg* | *SimRegArg* | *SimComboArg*

**get_value**(*state*, ***kwargs*)

**set_value**(*state*, *value*, ***kwargs*)

**class** angr.calling_conventions.**SimArrayArg**

Bases: *SimFunctionArgument*

**__init__**(*locs*)

**get_value**(*state*, ***kwargs*)

**set_value**(*state*, *value*, ***kwargs*)

**class** angr.calling_conventions.**SimReferenceArgument**

Bases: *SimFunctionArgument*

A function argument which is passed by reference.

**Variables**

- **ptr_loc** – The location the reference's pointer is stored
- **main_loc** – A SimStackArgument describing how to load the argument's value as if it were stored at offset zero on the stack. It will be passed `stack_base=ptr_loc.get_value(state)`

**__init__**(*ptr_loc*, *main_loc*)

**Parameters**

- **ptr_loc** (SimFunctionArgument)
- **main_loc** (SimFunctionArgument)

**get_value**(*state*, ***kwargs*)

**set_value**(*state*, *value*, ***kwargs*)

**class** angr.calling_conventions.**ArgSession**

Bases: object

A class to keep track of the state accumulated in laying parameters out into memory

**__init__**(*cc*)

**cc**

**fp_iter**

**int_iter**

**both_iter**

**getstate**()

**setstate**(*state*)

class angr.calling_conventions.**UsercallArgSession**

Bases: object

An argsession for use with SimCCUsercall

**__init__**(*cc*)

**cc**

**real_args**

**getstate**()

**setstate**(*state*)

class angr.calling_conventions.**SimCC**

Bases: object

A calling convention allows you to extract from a state the data passed from function to function by calls and returns. Most of the methods provided by SimCC that operate on a state assume that the program is just after a call but just before stack frame allocation, though this may be overridden with the *stack_base* parameter to each individual method.

This is the base class for all calling conventions.

**__init__**(*arch*)

**Parameters**
**arch** (Arch) – The Archinfo arch for this CC

**ARG_REGS**: list[str] = []

**FP_ARG_REGS**: list[str] = []

**STACKARG_SP_BUFF** = 0

**STACKARG_SP_DIFF** = 0

**CALLER_SAVED_REGS**: list[str] = []

**RETURN_ADDR**: *SimFunctionArgument* | None = None

**RETURN_VAL**: *SimFunctionArgument* | None = None

**OVERFLOW_RETURN_VAL**: *SimFunctionArgument* | None = None

**FP_RETURN_VAL**: *SimFunctionArgument* | None = None

**ARCH**: *type*[Arch] | None = None

**EXTRA_ARCHES**: tuple[*type*[Arch], ...] = ()

**CALLEE_CLEANUP** = False

**STACK_ALIGNMENT** = 1

**ARG_SLOT_SIZE**: int | None = None

**ARG_REG_SANITY_FILTER** = False

**STRICT_CALLER_SAVED_MATCH** = True

**classmethod arg_reg_offsets**(*arch*)

Every register file offset covered by this convention's argument registers.

**Return type**
set[int]

**Parameters**
**arch** (*Arch*)

**property arg_slot_size: int**

The width in bytes of one argument slot. See ARG_SLOT_SIZE.

**property int_args**

Iterate through all the possible arg positions that can only be used to store integer or pointer values.

Returns an iterator of SimFunctionArguments

**property memory_args**

Iterate through all the possible arg positions that can be used to store any kind of argument.

Returns an iterator of SimFunctionArguments

**property fp_args**

Iterate through all the possible arg positions that can only be used to store floating point values.

Returns an iterator of SimFunctionArguments

**is_fp_arg**(*arg*)

This should take a SimFunctionArgument instance and return whether or not that argument is a floating-point argument.

**Returns True for MUST be a floating point arg,**
False for MUST NOT be a floating point arg, None for when it can be either.

**class ArgSession**

Bases: object

A class to keep track of the state accumulated in laying parameters out into memory

**both_iter**

**cc**

**fp_iter**

**int_iter**

**__init__**(*cc*)

**getstate**()

**setstate**(*state*)

**arg_session**(*ret_ty*)

Return an arg session.

A session provides the control interface necessary to describe how integral and floating-point arguments are laid out into memory. The default behavior is that there are a finite list of int-only and fp-only argument slots, and an infinite number of generic slots, and when an argument of a given type is requested, the most slot available is used. If you need different behavior, subclass ArgSession.

You need to provide the return type of the function in order to kick off an arg layout session.

**Return type**
*ArgSession*

**Parameters**
**ret_ty** (SimType | *None*)

**return_in_implicit_outparam**(*ty*)

**Return type**
bool

**stack_space**(*args*)

**Parameters**
**args** – A list of SimFunctionArguments

**Returns**
The number of bytes that should be allocated on the stack to store all these args, NOT INCLUDING the return address.

**return_val**(*ty*, *perspective_returned=False*)

The location the return value is stored, based on its type.

**property return_addr**

The location the return address is stored.

**next_arg**(*session*, *arg_type*)

**Return type**
*SimFunctionArgument*

**Parameters**

- **session** (ArgSession)
- **arg_type** (SimType)

**static is_fp_value**(*val*)

**static guess_prototype**(*args*, *prototype=None*)

Come up with a plausible SimTypeFunction for the given args (as would be passed to e.g. setup_callsite).

You can pass a variadic function prototype in the *base_type* parameter and all its arguments will be used, only guessing types for the variadic arguments.

**arg_locs**(*prototype*)

**Return type**
list[*SimFunctionArgument*]

**get_args**(*state*, *prototype*, *stack_base=None*)

**set_return_val**(*state*, *val*, *ty*, *stack_base=None*, *perspective_returned=False*)

**setup_callsite**(*state*, *ret_addr*, *args*, *prototype*, *stack_base=None*, *alloc_base=None*, *grow_like_stack=True*)

This function performs the actions of the caller getting ready to jump into a function.

**Parameters**

- **state** – The SimState to operate on
- **ret_addr** – The address to return to when the called function finishes

- **args** – The list of arguments that that the called function will see
- **prototype** – The signature of the call you're making. Should include variadic args concretely.
- **stack_base** – An optional pointer to use as the top of the stack, circa the function entry point
- **alloc_base** – An optional pointer to use as the place to put excess argument data
- **grow_like_stack** – When allocating data at alloc_base, whether to allocate at decreasing addresses

The idea here is that you can provide almost any kind of python type in *args* and it'll be translated to a binary format to be placed into simulated memory. Lists (representing arrays) must be entirely elements of the same type and size, while tuples (representing structs) can be elements of any type and size. If you'd like there to be a pointer to a given value, wrap the value in a *PointerWrapper*.

If stack_base is not provided, the current stack pointer will be used, and it will be updated. If alloc_base is not provided, the stack base will be used and grow_like_stack will implicitly be True.

grow_like_stack controls the behavior of allocating data at alloc_base. When data from args needs to be wrapped in a pointer, the pointer needs to point somewhere, so that data is dumped into memory at alloc_base. If you set alloc_base to point to somewhere other than the stack, set grow_like_stack to False so that sequential allocations happen at increasing addresses.

**teardown_callsite**(*state*, *return_val=None*, *prototype=None*, *force_callee_cleanup=False*)

This function performs the actions of the callee as it's getting ready to return. It returns the address to return to.

**Parameters**

- **state** – The state to mutate
- **return_val** – The value to return
- **prototype** – The prototype of the given function
- **force_callee_cleanup** – If we should clean up the stack allocation for the arguments even if it's not the callee's job to do so

TODO: support the stack_base parameter from setup_callsite…? Does that make sense in this context? Maybe it could make sense by saying that you pass it in as something like the "saved base pointer" value?

**static find_cc**(*arch*, *args*, *sp_delta*, *platform='Linux'*, *unused_hint=None*, *extra_pop=None*, *language=None*)

Pinpoint the best-fit calling convention and return the corresponding SimCC instance, or None if no fit is found.

**Parameters**

- **arch** (Arch) – An ArchX instance. Can be obtained from archinfo.
- **args** (list[*SimRegArg* | *SimStackArg*]) – A list of arguments. It may be updated by the first matched calling convention to remove non-argument arguments.
- **sp_delta** (int) – The change of stack pointer before and after the call is made.
- **extra_pop** (int | None) – The number of bytes that are popped by the callee. This is used to distinguish between callee-cleanup and caller-cleanup conventions.
- **language** (str | None) – The source language of the binary (e.g. "go"), if known. Languages with their own ABI are matched against that ABI alone.

- **platform** (*str | None*)
- **unused_hint** (*list[SimRegArg] | None*)

**Return type**
*SimCC* | None

**Returns**
A calling convention instance, or None if none of the SimCC subclasses seems to fit the arguments provided.

**classmethod arches**()

**Return type**
tuple[type[Arch], ...]

**get_arg_info**(*state*, *prototype*)

This is just a simple wrapper that collects the information from various locations prototype is as passed to self.arg_locs and self.get_args :type angr.SimState state: :param angr.SimState state: The state to evaluate and extract the values from :return: A list of tuples, where the nth tuple is (type, name, location, value) of the nth argument

**class** angr.calling_conventions.**SimLyingRegArg**

Bases: *SimRegArg*

A register that LIES about the types it holds

**__init__**(*name*, *size=8*)

**get_value**(*state*, ***kwargs*)

**set_value**(*state*, *value*, ***kwargs*)

**refine**(*size*, *arch=None*, *offset=None*, *is_fp=None*)

**class** angr.calling_conventions.**SimCCUsercall**

Bases: *SimCC*

**ArgSession**

alias of *UsercallArgSession*

**next_arg**(*session*, *arg_type*)

**Parameters**
**session** (UsercallArgSession)

**class** angr.calling_conventions.**SimCCCdecl**

Bases: *SimCC*

**ARG_REGS: list[str] = []**

**FP_ARG_REGS: list[str] = []**

**STACKARG_SP_DIFF = 4**

**CALLER_SAVED_REGS: list[str] = ['eax', 'ecx', 'edx']**

**RETURN_VAL: *SimFunctionArgument* | None = <eax>**

**OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <edx>**

FP_RETURN_VAL: *SimFunctionArgument* | None = <st0>

RETURN_ADDR: *SimFunctionArgument* | None = [0x0]

ARCH
alias of ArchX86

next_arg(*session*, *arg_type*)

STRUCT_RETURN_THRESHOLD = 32

return_in_implicit_outparam(*ty*)

class angr.calling_conventions.SimCCMicrosoftCdecl
Bases: *SimCCCdecl*

STRUCT_RETURN_THRESHOLD = 64

class angr.calling_conventions.SimCCMicrosoftThiscall
Bases: *SimCCCdecl*

CALLEE_CLEANUP = True

ARG_REGS: list[str] = ['ecx']

CALLER_SAVED_REGS: list[str] = ['eax', 'ecx', 'edx']

STRUCT_RETURN_THRESHOLD = 64

arg_locs(*prototype*)

**Return type**
list[*SimFunctionArgument*]

class angr.calling_conventions.SimCCStdcall
Bases: *SimCCMicrosoftCdecl*

CALLEE_CLEANUP = True

class angr.calling_conventions.SimCCMicrosoftFastcall
Bases: *SimCC*

ARG_REGS: list[str] = ['ecx', 'edx']

CALLER_SAVED_REGS: list[str] = ['eax', 'ecx', 'edx']

STACKARG_SP_DIFF = 4

RETURN_VAL: *SimFunctionArgument* | None = <eax>

OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <edx>

RETURN_ADDR: *SimFunctionArgument* | None = [0x0]

ARCH
alias of ArchX86

next_arg(*session*, *arg_type*)

STRUCT_RETURN_THRESHOLD = 64

**return_in_implicit_outparam**(*ty*)

**class** angr.calling_conventions.**MicrosoftAMD64ArgSession**

Bases: *ArgSession*

**both_iter**

**cc**

**fp_iter**

**int_iter**

**class** angr.calling_conventions.**SimCCMicrosoftAMD64**

Bases: *SimCC*

**ARG_REGS: list[str] = ['rcx', 'rdx', 'r8', 'r9']**

**FP_ARG_REGS: list[str] = ['xmm0', 'xmm1', 'xmm2', 'xmm3']**

**CALLER_SAVED_REGS: list[str] = ['rax', 'rcx', 'rdx', 'r8', 'r9', 'r10', 'r11']**

**STACKARG_SP_DIFF = 8**

**STACKARG_SP_BUFF = 32**

**RETURN_VAL: *SimFunctionArgument* | None = <rax>**

**OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <rdx>**

**FP_RETURN_VAL: *SimFunctionArgument* | None = <xmm0>**

**RETURN_ADDR: *SimFunctionArgument* | None = [0x0]**

**ARCH**

alias of ArchAMD64

**STACK_ALIGNMENT = 16**

**ArgSession**

alias of *MicrosoftAMD64ArgSession*

**STRUCT_RETURN_THRESHOLD = 64**

**next_arg**(*session*, *arg_type*)

**return_in_implicit_outparam**(*ty*)

**class** angr.calling_conventions.**SimCCSyscall**

Bases: *SimCC*

The base class of all syscall CCs.

**ERROR_REG: *SimRegArg* = None**

**SYSCALL_ERRNO_START = None**

**static syscall_num**(*state*)

**Return type**
int

**linux_syscall_update_error_reg**(*state*, *expr*)

**set_return_val**(*state*, *val*, *ty*, ***kwargs*)

class angr.calling_conventions.**SimCCX86LinuxSyscall**

Bases: *SimCCSyscall*

**ARG_REGS: list[str] = ['ebx', 'ecx', 'edx', 'esi', 'edi', 'ebp']**

**FP_ARG_REGS: list[str] = []**

**RETURN_VAL: *SimFunctionArgument* | None = <eax>**

**RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>**

**ARCH**

alias of ArchX86

**static syscall_num**(*state*)

class angr.calling_conventions.**SimCCX86WindowsSyscall**

Bases: *SimCCSyscall*

**ARG_REGS: list[str] = ['ecx']**

**FP_ARG_REGS: list[str] = []**

**RETURN_VAL: *SimFunctionArgument* | None = <eax>**

**RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>**

**ARCH**

alias of ArchX86

**static syscall_num**(*state*)

class angr.calling_conventions.**SimCCSystemVAMD64**

Bases: *SimCC*

**ARG_REGS: list[str] = ['rdi', 'rsi', 'rdx', 'rcx', 'r8', 'r9']**

**FP_ARG_REGS: list[str] = ['xmm0', 'xmm1', 'xmm2', 'xmm3', 'xmm4', 'xmm5', 'xmm6', 'xmm7']**

**STACKARG_SP_DIFF = 8**

**CALLER_SAVED_REGS: list[str] = ['rdi', 'rsi', 'rdx', 'rcx', 'r8', 'r9', 'r10', 'r11', 'rax']**

**RETURN_ADDR: *SimFunctionArgument* | None = [0x0]**

**RETURN_VAL: *SimFunctionArgument* | None = <rax>**

**OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <rdx>**

**FP_RETURN_VAL: *SimFunctionArgument* | None = <xmm0>**

**OVERFLOW_FP_RETURN_VAL = <xmm1>**

**ARCH**

alias of ArchAMD64

**STACK_ALIGNMENT = 16**

**next_arg**(*session*, *arg_type*)

**return_in_implicit_outparam**(*ty*)

**class** angr.calling_conventions.**SimCCGoAMD64**

Bases: *SimCC*

Go's register-based internal ABI (ABIInternal), used by the gc toolchain on amd64 since go1.17.

Integer arguments and results are assigned to RAX, RBX, RCX, RDI, RSI, R8, R9, R10, R11 and floating-point ones to X0-X14. Results restart at the beginning of the same two sequences, so a multi-value return comes back in RAX, RBX, … – model it as a struct return type.

RDX carries the closure context pointer, R12/R13 are scratch, R14 pins the current goroutine (g) and X15 is a zero register; none of them are arguments. Unlike the System V ABI, Go assigns registers per scalar leaf rather than per eightbyte, and an aggregate that does not fit entirely in registers is passed wholly on the stack with the register counters rolled back.

**ARG_REGS: list[str] = ['rax', 'rbx', 'rcx', 'rdi', 'rsi', 'r8', 'r9', 'r10', 'r11']**

**FP_ARG_REGS: list[str] = ['xmm0', 'xmm1', 'xmm2', 'xmm3', 'xmm4', 'xmm5', 'xmm6', 'xmm7', 'xmm8', 'xmm9', 'xmm10', 'xmm11', 'xmm12', 'xmm13', 'xmm14']**

**STACKARG_SP_DIFF = 8**

**CALLER_SAVED_REGS: list[str] = ['rax', 'rbx', 'rcx', 'rdx', 'rdi', 'rsi', 'r8', 'r9', 'r10', 'r11', 'r12', 'r13', 'r15', 'xmm0', 'xmm1', 'xmm2', 'xmm3', 'xmm4', 'xmm5', 'xmm6', 'xmm7', 'xmm8', 'xmm9', 'xmm10', 'xmm11', 'xmm12', 'xmm13', 'xmm14']**

**RETURN_ADDR: *SimFunctionArgument* | None = [0x0]**

**RETURN_VAL: *SimFunctionArgument* | None = <rax>**

**OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <rbx>**

**FP_RETURN_VAL: *SimFunctionArgument* | None = <xmm0>**

**OVERFLOW_FP_RETURN_VAL = <xmm1>**

**ARCH**

alias of ArchAMD64

**STACK_ALIGNMENT = 8**

**ARG_REG_SANITY_FILTER = True**

**STRICT_CALLER_SAVED_MATCH = False**

**next_arg**(*session*, *arg_type*)

**Return type**
*SimFunctionArgument*

**Parameters**

- **session** (ArgSession)

- **arg_type** (SimType)

class angr.calling_conventions.**SimCCGoAMD64ABI0**

Bases: *SimCCGoAMD64*

Go's original all-stack ABI (ABI0), still used by the hand-written assembly in the runtime. Every argument and result is passed on the stack. The gc linker names these symbols "<name>.abi0".

**ARG_REGS**: list[str] = []

**FP_ARG_REGS**: list[str] = []

**CALLER_SAVED_REGS**: list[str] = ['rax', 'rbx', 'rcx', 'rdx', 'rdi', 'rsi', 'r8', 'r9', 'r10', 'r11', 'r12', 'r13', 'r15', 'xmm0', 'xmm1', 'xmm2', 'xmm3', 'xmm4', 'xmm5', 'xmm6', 'xmm7', 'xmm8', 'xmm9', 'xmm10', 'xmm11', 'xmm12', 'xmm13', 'xmm14']

**RETURN_VAL**: *SimFunctionArgument* | None = None

**OVERFLOW_RETURN_VAL**: *SimFunctionArgument* | None = None

**FP_RETURN_VAL**: *SimFunctionArgument* | None = None

**OVERFLOW_FP_RETURN_VAL** = None

class angr.calling_conventions.**SimCCAMD64LinuxSyscall**

Bases: *SimCCSyscall*

**ARG_REGS**: list[str] = ['rdi', 'rsi', 'rdx', 'r10', 'r8', 'r9']

**RETURN_VAL**: *SimFunctionArgument* | None = <rax>

**RETURN_ADDR**: *SimFunctionArgument* | None = <ip_at_syscall>

**ARCH**

alias of ArchAMD64

**CALLER_SAVED_REGS**: list[str] = ['rax', 'rcx', 'r11']

**static syscall_num**(*state*)

class angr.calling_conventions.**SimCCAMD64WindowsSyscall**

Bases: *SimCCSyscall*

**ARG_REGS**: list[str] = ['rcx']

**FP_ARG_REGS**: list[str] = []

**RETURN_VAL**: *SimFunctionArgument* | None = <rax>

**RETURN_ADDR**: *SimFunctionArgument* | None = <ip_at_syscall>

**ARCH**

alias of ArchAMD64

**static syscall_num**(*state*)

class angr.calling_conventions.**SimCCARM**

Bases: *SimCC*

**ARG_REGS**: list[str] = ['r0', 'r1', 'r2', 'r3']

FP_ARG_REGS: list[str] = []

CALLER_SAVED_REGS: list[str] = ['r0', 'r1', 'r2', 'r3']

RETURN_ADDR: *SimFunctionArgument* | None = <lr>

RETURN_VAL: *SimFunctionArgument* | None = <r0>

OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <r1>

ARCH

alias of ArchARM

next_arg(*session*, *arg_type*)

class angr.calling_conventions.SimCCARMHF

Bases: *SimCCARM*

ARG_REGS: list[str] = ['r0', 'r1', 'r2', 'r3']

FP_ARG_REGS: list[str] = ['s0', 's1', 's2', 's3', 's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14', 's15']

FP_RETURN_VAL: *SimFunctionArgument* | None = <s0>

CALLER_SAVED_REGS: list[str] = []

RETURN_ADDR: *SimFunctionArgument* | None = <lr>

RETURN_VAL: *SimFunctionArgument* | None = <r0>

OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <r1>

ARCH

alias of ArchARMHF

EXTRA_ARCHES: tuple[*type*[Arch], ...] = (<class 'archinfo.arch_arm.ArchARMCortexM'>,)

next_arg(*session*, *arg_type*)

**Parameters**

**session** (ArgSession)

class angr.calling_conventions.SimCCARMLinuxSyscall

Bases: *SimCCSyscall*

ARG_REGS: list[str] = ['r0', 'r1', 'r2', 'r3']

FP_ARG_REGS: list[str] = []

RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>

RETURN_VAL: *SimFunctionArgument* | None = <r0>

ARCH

alias of ArchARM

static syscall_num(*state*)

class angr.calling_conventions.**SimCCARMWindowsSyscall**

Bases: *SimCCSyscall*

The system call convention of Windows on 32-bit ARM.

An ntdll stub loads the service index into r12 and traps with svc #1; arguments follow the ordinary ARM convention. See https://codemachine.com/articles/arm_assembler_primer.html.

**ARG_REGS**: list[str] = ['r0', 'r1', 'r2', 'r3']

**FP_ARG_REGS**: list[str] = []

**RETURN_ADDR**: *SimFunctionArgument* | None = <ip_at_syscall>

**RETURN_VAL**: *SimFunctionArgument* | None = <r0>

**ARCH**

alias of ArchARM

**static syscall_num**(*state*)

class angr.calling_conventions.**SimCCAArch64**

Bases: *SimCC*

**ARG_REGS**: list[str] = ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7']

**FP_ARG_REGS**: list[str] = []

**RETURN_ADDR**: *SimFunctionArgument* | None = <lr>

**RETURN_VAL**: *SimFunctionArgument* | None = <x0>

**ARCH**

alias of ArchAArch64

class angr.calling_conventions.**SimCCAArch64LinuxSyscall**

Bases: *SimCCSyscall*

**ARG_REGS**: list[str] = ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7']

**FP_ARG_REGS**: list[str] = []

**RETURN_VAL**: *SimFunctionArgument* | None = <x0>

**RETURN_ADDR**: *SimFunctionArgument* | None = <ip_at_syscall>

**ARCH**

alias of ArchAArch64

**static syscall_num**(*state*)

class angr.calling_conventions.**SimCCAArch64WindowsSyscall**

Bases: *SimCCSyscall*

The system call convention of Windows on AArch64.

An ntdll stub is a bare "svc #index; ret": Windows encodes the service index in the SVC immediate rather than in a register, and the kernel recovers it from ESR_EL1 as the low 12 bits. Arguments follow the ordinary AArch64 convention. See https://gracefulbits.wordpress.com/2018/07/26/system-call-dispatching-for-windows-on-arm64/.

ARG_REGS: list[str] = ['x0', 'x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7']

FP_ARG_REGS: list[str] = []

RETURN_VAL: *SimFunctionArgument* | None = <x0>

RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>

ARCH
alias of ArchAArch64

static syscall_num(*state*)

class angr.calling_conventions.SimCCRISCV64
Bases: *SimCC*

ARG_REGS: list[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

FP_ARG_REGS: list[str] = ['fa0', 'fa1', 'fa2', 'fa3', 'fa4', 'fa5', 'fa6', 'fa7']

FP_RETURN_VAL: *SimFunctionArgument* | None = <fa0>

RETURN_VAL: *SimFunctionArgument* | None = <a0>

RETURN_ADDR: *SimFunctionArgument* | None = <ra>

CALLER_SAVED_REGS: list[str] = ['ra', 'a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7', 't0', 't1', 't2', 't3', 't4', 't5', 't6']

STACK_ALIGNMENT = 16

ARCH
alias of ArchRISCV64

next_arg(*session*, *arg_type*)

class angr.calling_conventions.SimCCRISCV64LinuxSyscall
Bases: *SimCCSyscall*

ARG_REGS: list[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5']

FP_ARG_REGS: list[str] = []

RETURN_VAL: *SimFunctionArgument* | None = <a0>

RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>

ARCH
alias of ArchRISCV64

static syscall_num(*state*)

class angr.calling_conventions.SimCCO32
Bases: *SimCC*

ARG_REGS: list[str] = ['a0', 'a1', 'a2', 'a3']

FP_ARG_REGS: list[str] = ['f12', 'f13', 'f14', 'f15']

STACKARG_SP_BUFF = 16

CALLER_SAVED_REGS: list[str] = ['t9', 'gp']

RETURN_ADDR: *SimFunctionArgument* | None = <ra>

RETURN_VAL: *SimFunctionArgument* | None = <v0>

OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <v1>

ARCH
alias of ArchMIPS32

next_arg(*session*, *arg_type*)

class angr.calling_conventions.SimCCO32LinuxSyscall
Bases: *SimCCSyscall*

ARG_REGS: list[str] = ['a0', 'a1', 'a2', 'a3']

FP_ARG_REGS: list[str] = []

RETURN_VAL: *SimFunctionArgument* | None = <v0>

RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>

ARCH
alias of ArchMIPS32

ERROR_REG: *SimRegArg* = <a3>

SYSCALL_ERRNO_START = -1133

static syscall_num(*state*)

class angr.calling_conventions.SimCCN64
Bases: *SimCC*

ARG_REGS: list[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

CALLER_SAVED_REGS: list[str] = ['t9', 'gp']

FP_ARG_REGS: list[str] = []

STACKARG_SP_BUFF = 32

RETURN_ADDR: *SimFunctionArgument* | None = <ra>

RETURN_VAL: *SimFunctionArgument* | None = <v0>

ARCH
alias of ArchMIPS64

angr.calling_conventions.SimCCO64
alias of *SimCCN64*

class angr.calling_conventions.SimCCN32
Bases: *SimCCN64*

n32 passes its arguments like n64 does and differs only in the width of a pointer.

ARCH
alias of ArchMIPSN32

ARG_SLOT_SIZE: int | None = 8

class angr.calling_conventions.SimCCN64LinuxSyscall

Bases: *SimCCSyscall*

ARG_REGS: list[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

FP_ARG_REGS: list[str] = []

RETURN_VAL: *SimFunctionArgument* | None = <v0>

RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>

ARCH

alias of ArchMIPS64

ERROR_REG: *SimRegArg* = <a3>

SYSCALL_ERRNO_START = -1133

static syscall_num(*state*)

class angr.calling_conventions.SimCCN32LinuxSyscall

Bases: *SimCCN64LinuxSyscall*

ARCH

alias of ArchMIPSN32

ARG_SLOT_SIZE: int | None = 8

class angr.calling_conventions.SimCCPowerPC

Bases: *SimCC*

ARG_REGS: list[str] = ['r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10']

FP_ARG_REGS: list[str] = []

STACKARG_SP_BUFF = 8

RETURN_ADDR: *SimFunctionArgument* | None = <lr>

RETURN_VAL: *SimFunctionArgument* | None = <r3>

OVERFLOW_RETURN_VAL: *SimFunctionArgument* | None = <r4>

ARCH

alias of ArchPPC32

class angr.calling_conventions.SimCCPowerPCLinuxSyscall

Bases: *SimCCSyscall*

ARG_REGS: list[str] = ['r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10']

FP_ARG_REGS: list[str] = []

RETURN_VAL: *SimFunctionArgument* | None = <r3>

RETURN_ADDR: *SimFunctionArgument* | None = <ip_at_syscall>

**ARCH**

alias of ArchPPC32

**ERROR_REG: *SimRegArg*** = <cr0_0>

**SYSCALL_ERRNO_START** = -515

**static syscall_num**(*state*)

**class** angr.calling_conventions.**SimCCPowerPC64**

Bases: *SimCC*

**ARG_REGS: list[str]** = ['r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10']

**FP_ARG_REGS: list[str]** = []

**STACKARG_SP_BUFF** = 112

**RETURN_ADDR: *SimFunctionArgument* | None** = <lr>

**RETURN_VAL: *SimFunctionArgument* | None** = <r3>

**ARCH**

alias of ArchPPC64

**class** angr.calling_conventions.**SimCCPowerPC64LinuxSyscall**

Bases: *SimCCSyscall*

**ARG_REGS: list[str]** = ['r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10']

**FP_ARG_REGS: list[str]** = []

**RETURN_VAL: *SimFunctionArgument* | None** = <r3>

**RETURN_ADDR: *SimFunctionArgument* | None** = <ip_at_syscall>

**ARCH**

alias of ArchPPC64

**ERROR_REG: *SimRegArg*** = <cr0_0>

**SYSCALL_ERRNO_START** = -515

**static syscall_num**(*state*)

**class** angr.calling_conventions.**SimCCSoot**

Bases: *SimCC*

**ARCH**

alias of ArchSoot

**ARG_REGS: list[str]** = []

**class** angr.calling_conventions.**SimCCUnknown**

Bases: *SimCC*

Represent an unknown calling convention.

**class** angr.calling_conventions.**SimCCS390X**

Bases: *SimCC*

**ARG_REGS**: list[str] = ['r2', 'r3', 'r4', 'r5', 'r6']

**FP_ARG_REGS**: list[str] = ['f0', 'f2', 'f4', 'f6']

**STACKARG_SP_BUFF** = 160

**RETURN_ADDR**: *SimFunctionArgument* | None = <r14>

**RETURN_VAL**: *SimFunctionArgument* | None = <r2>

**ARCH**

alias of ArchS390X

class angr.calling_conventions.**SimCCS390XLinuxSyscall**

Bases: *SimCCSyscall*

**ARG_REGS**: list[str] = ['r2', 'r3', 'r4', 'r5', 'r6', 'r7']

**FP_ARG_REGS**: list[str] = []

**RETURN_VAL**: *SimFunctionArgument* | None = <r2>

**RETURN_ADDR**: *SimFunctionArgument* | None = <ip_at_syscall>

**ARCH**

alias of ArchS390X

**static syscall_num**(*state*)

angr.calling_conventions.**register_default_cc_by_language**(*language*, *arch*, *cc*, *platform='Linux'*)

**Parameters**

- **language** (*str*)
- **arch** (*str*)
- **cc** (type[SimCC])
- **platform** (*str*)

angr.calling_conventions.**register_default_cc**(*arch*, *cc*, *platform='Linux'*)

**Parameters**

- **arch** (*str*)
- **cc** (type[SimCC])
- **platform** (*str*)

angr.calling_conventions.**default_cc**(*arch*, *platform='Linux'*, *language=None*, *syscall=False*, *default=None*)

Return the default calling convention for a given architecture, platform, and language combination.

**Parameters**

- **arch** (str) – The architecture name.
- **platform** (str | None) – The platform name (e.g., “Linux” or “Win32”).
- **language** (str | None) – The programming language name (e.g., “go”).
- **syscall** (bool) – Return syscall convention (True), or normal calling convention (False, default).

- **default** (type[*SimCC*] | None) – The default calling convention to return if nothing fits.

**Return type**
type[*SimCC*] | None

**Returns**
A default calling convention class if we can find one for the architecture, platform, and language combination, or the default if nothing fits.

angr.calling_conventions.**project_language**(*project*)

The source language to pick a calling convention for, or None when there is no confident answer.

A language-specific ABI reinterprets every argument and return value in the binary, so a detection that rests on something as thin as a single matching symbol name is not acted upon.

**Return type**
str | None

angr.calling_conventions.**default_cc_for_project**(*project*, *syscall=False*, *default=None*)

Return the default calling convention for a project, taking its architecture, platform and detected source language into account.

**Return type**
type[*SimCC*] | None

**Parameters**

- **syscall** (*bool*)
- **default** (*type[SimCC] | None*)

angr.calling_conventions.**unify_arch_name**(*arch*)

Return the unified architecture name.

**Parameters**
**arch** (str) – The architecture name.

**Return type**
str

**Returns**
A unified architecture name.

angr.calling_conventions.**register_syscall_cc**(*arch*, *os*, *cc*)

angr.calling_conventions.**cls_**

alias of *SimCCUsercall*

angr.calling_conventions.**is_stack_probe**(*func*)

Return True if `func` is a Windows stack-probe / alloca-probe helper (`__chkstk` / `__alloca_probe`).

**Return type**
bool

**Parameters**
**func** (Function)

angr.calling_conventions.**call_clobbered_regs**(*cc*, *target*, *arch*)

Return the caller-saved (clobbered) register names for a call to `target` under calling convention `cc`.

This function is aware of special behaviors of functions, such as stack-probe callees (`__chkstk` / `__alloca_probe`). Future logic for other special functions may be added to this function as needed.

**Return type**
list[str]

**Parameters**

- **cc** (SimCC)
- **target** (Function | *None*)
- **arch** (*Arch*)

## 10.10 angr.code_location

**class** angr.code_location.**CodeLocation**

Bases: Generic

Stands for a specific program point by specifying basic block address and statement ID (for IRSBs), or SimProcedure name (for SimProcedures).

**__init__**(*block_addr*, *stmt_idx*, *sim_procedure=None*, *ins_addr=None*, *context=None*, *block_idx=None*, ***kwargs*)

Constructor.

**Parameters**

- **block_addr** (TypeVar(BlockAddr, bound= int | None)) – Address of the block
- **stmt_idx** (TypeVar(StmtIdx, bound= int | None)) – Statement ID. None for SimProcedures or if the code location is meant to refer to the entire block.
- **sim_procedure** (*class*) – The corresponding SimProcedure class.
- **ins_addr** (int | None) – The instruction address.
- **context** (TypeVar(Context)) – A tuple that represents the context of this CodeLocation in contextual mode, or None in contextless mode.
- **kwargs** – Optional arguments, will be stored, but not used in __eq__ or __hash__.
- **block_idx** (*int* | *None*)

**block_addr**

**stmt_idx**

**sim_procedure**

**ins_addr**

**context**

**block_idx**

**info: dict[str, Any] | None**

**property short_repr**

**class** angr.code_location.**ExternalCodeLocation**

Bases: *CodeLocation*

Stands for a program point that originates from outside an analysis' scope. i.e. a value loaded from rdi in a callee where the caller has not been analyzed.

call_string

class angr.code_location.**AILCodeLocation**

Bases: object

A code location that refers precisely to a statement of an AIL block, with an optional instruction address.

addr: int

block_idx: int | None

stmt_idx: int

insn_addr: int | None = None

static make_extern(*idx*)

**Parameters**
**idx** (*int*)

property insn_addr_unwrap: int

property is_extern: bool

property extern_idx_unwrap: int

property extern_idx: int | None

static from_codeloc(*codeloc*)

**Return type**
*AILCodeLocation*

**Parameters**
**codeloc** (CodeLocation)

property bbl_addr

property block_addr

property ins_addr

__init__(*addr*, *block_idx*, *stmt_idx*, *insn_addr=None*)

**Parameters**

- **addr** (*int*)
- **block_idx** (*int | None*)
- **stmt_idx** (*int*)
- **insn_addr** (*int | None*)

**Return type**
None

## 10.11 angr.codenode

angr.codenode.**repr_addr**(*addr*)

**Return type**
str

**Parameters**
**addr** (*K*)

**class** angr.codenode.**CodeNode**

Bases: Generic

The base class of nodes in a function graph.

**__init__**(*addr*, *size*, *graph=None*, *thumb=False*)

**Parameters**

- **addr** (*K*)
- **size** (*int*)

**addr**

**size: int**

**thumb**

**set_graph**(*graph*)

**successors**()

**Return type**
list[*CodeNode*]

**predecessors**()

**is_hook = None**

**class** angr.codenode.**BlockNode**

Bases: *CodeNode*, Generic

Represents a block of code in a function graph.

**is_hook = False**

**__init__**(*addr*, *size*, *bytestr=None*, ***kwargs*)

**Parameters**
**addr** (*int*)

**bytestr**

**class** angr.codenode.**SootBlockNode**

Bases: *BlockNode*[SootMethodDescriptor]

Represents a Soot block of code in a function graph.

**__init__**(*addr*, *size*, *stmts*, ***kwargs*)

**Parameters**
**addr** (*SootMethodDescriptor*)

**stmts**

**class** angr.codenode.**FuncNode**

Bases: *CodeNode*, Generic

Represents a function callee in a function graph.

**__init__**(*addr*, *func_name=None*, ***kwargs*)

**Parameters**

- **addr** (*K*)
- **func_name** (*str | None*)

**func_name**

**property is_addr_known: bool**

**class** angr.codenode.**HookNode**

Bases: *CodeNode*, Generic

Represents a hook in a function graph.

**is_hook = True**

**__init__**(*addr*, *size*, *sim_procedure*, ***kwargs*)

**Parameters**

**sim_procedure** (*SimProcedure* | None) – the the sim_procedure class

**sim_procedure**

**class** angr.codenode.**SyscallNode**

Bases: *HookNode*, Generic

Represents a syscall in a function graph.

**is_hook = False**

**sim_procedure**

## 10.12 angr.concretization_strategies

**class** angr.concretization_strategies.**SimConcretizationStrategy**

Bases: object

Concretization strategies control the resolution of symbolic memory indices in SimuVEX. By subclassing this class and setting it as a concretization strategy (on state.memory.read_strategies and state.memory.write_strategies), SimuVEX's memory index concretization behavior can be modified.

**__init__**(*filter=None*, *exact=True*)

Initializes the base SimConcretizationStrategy.

**Parameters**

- **filter** – A function, taking arguments of (SimMemory, claripy.AST) that determines if this strategy can handle resolving the provided AST.
- **exact** – A flag (default: True) that determines if the convenience resolution functions provided by this class use exact or approximate resolution.

**concretize**(*memory*, *addr*, ***kwargs*)

Concretizes the address into a list of values. If this strategy cannot handle this address, returns None.

**copy**()

Returns a copy of the strategy, if there is data that should be kept separate between states. If not, returns self.

**merge**(*others*)

Merges this strategy with others (if there is data that should be kept separate between states. If not, is a no-op.

**class** angr.concretization_strategies.**SimConcretizationStrategyAny**

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any single solution.

**class** angr.concretization_strategies.**SimConcretizationStrategyControlledData**

Bases: *SimConcretizationStrategy*

Concretization strategy that constraints the address to controlled data. Controlled data consists of symbolic data and the addresses given as arguments. memory.

**class** angr.concretization_strategies.**SimConcretizationStrategyEval**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves an address into some limited number of solutions. Always handles the concretization, but only returns a maximum of limit number of solutions. Therefore, should only be used as the fallback strategy.

**class** angr.concretization_strategies.**SimConcretizationStrategyMax**

Bases: *SimConcretizationStrategy*

Concretization strategy that returns the maximum address.

**class** angr.concretization_strategies.**SimConcretizationStrategyNonzero**

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any non-zero solution.

**class** angr.concretization_strategies.**SimConcretizationStrategyNonzeroRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range in a non-zero location.

**class** angr.concretization_strategies.**SimConcretizationStrategyNorepeats**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses, without repeating.

**class** angr.concretization_strategies.**SimConcretizationStrategyNorepeatsRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range, with no repeats.

**class** angr.concretization_strategies.**SimConcretizationStrategyRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range.

**class** angr.concretization_strategies.**SimConcretizationStrategySingle**

Bases: *SimConcretizationStrategy*

Concretization strategy that ensures a single solution for an address.

**class** angr.concretization_strategies.**SimConcretizationStrategySolutions**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves an address into some limited number of solutions.

**class** angr.concretization_strategies.**SimConcretizationStrategyUnlimitedRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range without checking if the number of possible addresses is within the limit.

### Submodules


*any*
*any_named*
*base*
*controlled_data*
*eval*
*logging*
*max*
*nonzero*
*nonzero_range*
*norepeats*
*norepeats_range*
*range*
*signed_add*
*single*
*solutions*
*unlimited_range*


## 10.12.1 angr.concretization_strategies.any

**class** angr.concretization_strategies.any.**SimConcretizationStrategyAny**

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any single solution.

## 10.12.2 angr.concretization_strategies.any_named

**class** angr.concretization_strategies.any_named.**SimConcretizationStrategyAnyNamed**

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any single solution and creates a BVS at the resulting address.

## 10.12.3 angr.concretization_strategies.base

**class** angr.concretization_strategies.base.**SimConcretizationStrategy**

Bases: object

Concretization strategies control the resolution of symbolic memory indices in SimuVEX. By subclassing this class and setting it as a concretization strategy (on state.memory.read_strategies and state.memory.write_strategies), SimuVEX's memory index concretization behavior can be modified.

**__init__**(*filter=None*, *exact=True*)

Initializes the base SimConcretizationStrategy.

**Parameters**

- **filter** – A function, taking arguments of (SimMemory, claripy.AST) that determines if this strategy can handle resolving the provided AST.
- **exact** – A flag (default: True) that determines if the convenience resolution functions provided by this class use exact or approximate resolution.

**concretize**(*memory*, *addr*, ***kwargs*)

Concretizes the address into a list of values. If this strategy cannot handle this address, returns None.

**copy**()

Returns a copy of the strategy, if there is data that should be kept separate between states. If not, returns self.

**merge**(*others*)

Merges this strategy with others (if there is data that should be kept separate between states. If not, is a no-op.

### 10.12.4 angr.concretization_strategies.controlled_data

**class** angr.concretization_strategies.controlled_data.**SimConcretizationStrategyControlledData**

Bases: *SimConcretizationStrategy*

Concretization strategy that constraints the address to controlled data. Controlled data consists of symbolic data and the addresses given as arguments. memory.

### 10.12.5 angr.concretization_strategies.eval

**class** angr.concretization_strategies.eval.**SimConcretizationStrategyEval**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves an address into some limited number of solutions. Always handles the concretization, but only returns a maximum of limit number of solutions. Therefore, should only be used as the fallback strategy.

### 10.12.6 angr.concretization_strategies.logging

**class** angr.concretization_strategies.logging.**SimConcretizationStrategyLogging**

Bases: *SimConcretizationStrategy*

Concretization strategy that logs concretization results from another strategy.

### 10.12.7 angr.concretization_strategies.max

**class** angr.concretization_strategies.max.**SimConcretizationStrategyMax**

Bases: *SimConcretizationStrategy*

Concretization strategy that returns the maximum address.

### 10.12.8 angr.concretization_strategies.nonzero

**class** angr.concretization_strategies.nonzero.**SimConcretizationStrategyNonzero**

Bases: *SimConcretizationStrategy*

Concretization strategy that returns any non-zero solution.

### 10.12.9 angr.concretization_strategies.nonzero_range

**class** angr.concretization_strategies.nonzero_range.**SimConcretizationStrategyNonzeroRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range in a non-zero location.

### 10.12.10 angr.concretization_strategies.norepeats

**class** angr.concretization_strategies.norepeats.**SimConcretizationStrategyNorepeats**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses, without repeating.

### 10.12.11 angr.concretization_strategies.norepeats_range

**class** angr.concretization_strategies.norepeats_range.**SimConcretizationStrategyNorepeatsRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves a range, with no repeats.

### 10.12.12 angr.concretization_strategies.range

**class** angr.concretization_strategies.range.**SimConcretizationStrategyRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range.

### 10.12.13 angr.concretization_strategies.signed_add

**class** angr.concretization_strategies.signed_add.**SimConcretizationStrategySignedAdd**

Bases: *SimConcretizationStrategy*

Concretization strategy that changes additions of big offsets to subtractions of small offsets.

### 10.12.14 angr.concretization_strategies.single

**class** angr.concretization_strategies.single.**SimConcretizationStrategySingle**

Bases: *SimConcretizationStrategy*

Concretization strategy that ensures a single solution for an address.

### 10.12.15 angr.concretization_strategies.solutions

**class** angr.concretization_strategies.solutions.**SimConcretizationStrategySolutions**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves an address into some limited number of solutions.

### 10.12.16 angr.concretization_strategies.unlimited_range

**class** angr.concretization_strategies.unlimited_range.**SimConcretizationStrategyUnlimitedRange**

Bases: *SimConcretizationStrategy*

Concretization strategy that resolves addresses to a range without checking if the number of possible addresses is within the limit.

## 10.13 angr.distributed

angr.distributed provides a simple implementation for conducting long-running symbolic-execution-based tasks.

**class** angr.distributed.**Server**

Bases: object

Server implements the analysis server with a series of control interfaces exposed.

**Variables**

- **project** – An instance of angr.Project.
- **spill_yard** (*str*) – A directory to store spilled states.
- **db** (*str*) – Path of the database that stores information about spilled states.
- **max_workers** (*int*) – Maximum number of workers. Each worker starts a new process.
- **max_states** (*int*) – Maximum number of active states for each worker.
- **staging_max** (*int*) – Maximum number of inactive states that are kept into memory before spilled onto the disk and potentially be picked up by another worker.
- **bucketizer** (*bool*) – Use the Bucketizer exploration strategy.
- **_worker_exit_callback** – A method that will be called upon the exit of each worker.

**__init__**(*project*, *spill_yard=None*, *db=None*, *max_workers=None*, *max_states=10*, *staging_max=10*, *bucketizer=True*, *recursion_limit=1000*, *worker_exit_callback=None*, *techniques=None*, *add_options=None*, *remove_options=None*)

**inc_active_workers**()

**dec_active_workers**()

**stop**()

**property active_workers**

**property stopped**

**on_worker_exit**(*worker_id*, *stashes*)

**run**()

#### Submodules

*server*
*worker*

### 10.13.1 angr.distributed.server

**class** angr.distributed.server.**Server**

Bases: object

Server implements the analysis server with a series of control interfaces exposed.

**Variables**

- **project** – An instance of angr.Project.
- **spill_yard** (*str*) – A directory to store spilled states.
- **db** (*str*) – Path of the database that stores information about spilled states.
- **max_workers** (*int*) – Maximum number of workers. Each worker starts a new process.
- **max_states** (*int*) – Maximum number of active states for each worker.
- **staging_max** (*int*) – Maximum number of inactive states that are kept into memory before spilled onto the disk and potentially be picked up by another worker.
- **bucketizer** (*bool*) – Use the Bucketizer exploration strategy.
- **_worker_exit_callback** – A method that will be called upon the exit of each worker.

**__init__**(*project*, *spill_yard=None*, *db=None*, *max_workers=None*, *max_states=10*, *staging_max=10*, *bucketizer=True*, *recursion_limit=1000*, *worker_exit_callback=None*, *techniques=None*, *add_options=None*, *remove_options=None*)

**inc_active_workers**()

**dec_active_workers**()

**stop**()

**property active_workers**

**property stopped**

**on_worker_exit**(*worker_id*, *stashes*)

**run**()

### 10.13.2 angr.distributed.worker

**class** angr.distributed.worker.**BadStatesDropper**

Bases: *ExplorationTechnique*

Dumps and drops states that are not "active".

**__init__**(*vault*, *db*)

**class** angr.distributed.worker.**ExplorationStatusNotifier**

Bases: *ExplorationTechnique*

Force the exploration to stop if the server.stop is True.

**__init__**(*server_state*)

**Parameters**

**server_state** (*dict*)

class angr.distributed.worker.**Worker**

Bases: object

Worker implements a worker thread/process for conducting a task.

**__init__**(*worker_id*, *server*, *server_state*, *recursion_limit=None*, *techniques=None*, *add_options=None*, *remove_options=None*)

**start**()

**run**(*initializer*)

**Parameters**
**initializer** (Initializer)

## 10.14 angr.emulator

class angr.emulator.**Emulator**

Bases: object

Emulator is a utility that adapts an angr *SuccessorsEngine* to a more user-friendly interface for concrete execution. It only supports concrete execution.

Saftey: This class is not thread-safe. It should only be used in a single-threaded context. It can be safely shared between multiple threads, provided that only one thread is using it at a time.

**__init__**(*engine*, *init_state*)

**Parameters**

- **engine** (*SuccessorsEngine*) – The *SuccessorsEngine* to use for emulation.
- **init_state** (*SimState*) – The initial state to use for emulation.

**property state:** *SimState*

The current state of the emulator.

**property breakpoints:** set[int]

The set of currently set breakpoints.

**add_breakpoint**(*addr*)

Add a breakpoint at the given address.

**Parameters**
**addr** (int) – The address to set the breakpoint at.

**Return type**
None

**remove_breakpoint**(*addr*)

Remove a breakpoint at the given address, if present.

**Parameters**
**addr** (int) – The address to remove the breakpoint from.

**Return type**
None

**run**(*num_inst=None*)

Execute the emulator.

**Return type**
*EmulatorStopReason*

**Parameters**
**num_inst** (*int | None*)

**exception** angr.emulator.**EmulatorException**

Bases: *AngrError*

Base class for exceptions raised by the Emulator.

**class** angr.emulator.**EmulatorStopReason**

Bases: Enum

Enum representing the reason for stopping the emulator.

**INSTRUCTION_LIMIT = 'instruction_limit'**

**BREAKPOINT = 'breakpoint'**

**NO_SUCCESSORS = 'no_successors'**

**MEMORY_ERROR = 'memory_error'**

**FAILURE = 'failure'**

**EXIT = 'exit'**

**exception** angr.emulator.**EngineException**

Bases: *EmulatorException*

Exception raised when the emulator encounters an unhandlable error in the engine.

**exception** angr.emulator.**StateDivergedException**

Bases: *EmulatorException*

Exception raised when an engine returns multiple successors.

## 10.15 angr.engines

**class** angr.engines.**HeavyResilienceMixin**

Bases: *VEXResilienceMixin*, *ClaripyDataMixin*

**class** angr.engines.**HeavyVEXMixin**

Bases: *SuccessorsEngine*, *ClaripyDataMixin*, *SimStateStorageMixin*, *VEXMixin*, *VEXLifter*

Execution engine based on VEX, Valgrind's IR.

Responds to the following parameters to the step stack:

- irsb: The PyVEX IRSB object to use for execution. If not provided one will be lifted.
- skip_stmts: The number of statements to skip in processing
- last_stmt: Do not execute any statements after this statement
- whitelist: Only execute statements in this set
- thumb: Whether the block should be force to be lifted in ARM's THUMB mode.

- **extra_stop_points:**
  An extra set of points at which to break basic blocks
- opt_level: The VEX optimization level to use.
- insn_bytes: A string of bytes to use for the block instead of the project.
- size: The maximum size of the block, in bytes.
- num_inst: The maximum number of instructions.
- traceflags: traceflags to be passed to VEX. (default: 0)

**class** angr.engines.**HooksMixin**

Bases: *SuccessorsEngine*, *ProcedureMixin*

A SimEngine mixin which adds a SimSuccessors handler which will look into the project's hooks and run the hook at the current address.

Will respond to the following parameters provided to the step stack:

- procedure: A SimProcedure instance to force-run instead of consulting the current hooks
- ret_to: An address to force-return-to at the end of the procedure

**class** angr.engines.**ProcedureEngine**

Bases: *ProcedureMixin*, *SuccessorsEngine*

A SimEngine that you may use if you only care about processing SimProcedures. *Requires* the procedure kwarg to be passed to process.

**class** angr.engines.**ProcedureMixin**

Bases: object

A mixin for SimEngine which adds the `process_procedure` method for calling a SimProcedure and adding its results to a SimSuccessors.

**process_procedure**(*state*, *successors*, *procedure*, *ret_to=None*, *arguments=None*, ***kwargs*)

**class** angr.engines.**SimEngine**

Bases: Generic

A SimEngine is a type which understands how to perform execution on a state.

**state:** *StateType*

**__init__**(*project*)

**Parameters**
**project** (Project)

**class** angr.engines.**SimEngineFailure**

Bases: *SuccessorsEngine*, *ProcedureMixin*

**class** angr.engines.**SimEngineSyscall**

Bases: *SuccessorsEngine*, *ProcedureMixin*

A SimEngine mixin which adds a successors handling step that checks if a syscall was just requested and if so handles it as a step.

class angr.engines.**SimEngineUnicorn**

Bases: *SuccessorsEngine*

Concrete execution in the Unicorn Engine, a fork of qemu.

Responds to the following parameters in the step stack:

- step: How many basic blocks we want to execute
- extra_stop_points: A collection of addresses at which execution should halt

**__init__**(*project*)

**Parameters**
**project** (Project)

class angr.engines.**SimInspectMixin**

Bases: *VEXMixin*

**handle_vex_block**(*irsb*)

class angr.engines.**SimSuccessors**

Bases: object

This class serves as a categorization of all the kinds of result states that can come from a SimEngine run.

**Variables**

- **addr** (*int*) – The address at which execution is taking place, as a python int
- **initial_state** – The initial state for which execution produced these successors
- **engine** – The engine that produced these successors
- **sort** (str | None) – A string identifying the type of engine that produced these successors
- **processed** (*bool*) – Whether or not the processing succeeded
- **description** (*str*) – A textual description of the execution step

The successor states produced by this run are categorized into several lists:

**Variables**

- **artifacts** (*dict*) – Any analysis byproducts (for example, an IRSB) that were produced during execution
- **successors** (list[*HeavyState*]) – The "normal" successors. IP may be symbolic, but must have reasonable number of solutions
- **unsat_successors** (list[*HeavyState*]) – Any successor which is unsatisfiable after its guard condition is added.
- **all_successors** (list[*HeavyState*]) – successors + unsat_successors
- **flat_successors** (list[*HeavyState*]) – The normal successors, but any symbolic IPs have been concretized. There is one state in this list for each possible value an IP may be concretized to for each successor state.
- **unconstrained_successors** (list[*HeavyState*]) – Any state for which during the flattening process we find too many solutions.

A more detailed description of the successor lists may be found here: https://docs.angr.io/core-concepts/simulation#simsuccessors

**__init__**(*addr*, *initial_state*)

**Parameters**

- **addr** (*int* | *SootAddressDescriptor* | *None*)
- **initial_state** (HeavyState | *None*)

**property is_empty**

**add_successor**(*state*, *target*, *guard*, *jumpkind*, *add_guard=True*, *exit_stmt_idx=None*, *exit_ins_addr=None*, *source=None*)

Add a successor state of the SimRun. This procedure stores method parameters into state.scratch, does some housekeeping, and calls out to helper functions to prepare the state and categorize it into the appropriate successor lists.

**Parameters**

- **state** (SimState) – The successor state.
- **target** – The target (of the jump/call/ret).
- **guard** – The guard expression.
- **jumpkind** (*str*) – The jumpkind (call, ret, jump, or whatnot).
- **add_guard** (*bool*) – Whether to add the guard constraint (default: True).
- **exit_stmt_idx** (*int*) – The ID of the exit statement, an integer by default. 'default' stands for the default exit, and None means it's not from a statement (for example, from a SimProcedure).
- **exit_ins_addr** (*int*) – The instruction pointer of this exit, which is an integer by default.
- **source** (*int*) – The source of the jump (i.e., the address of the basic block).

**class** angr.engines.**SootMixin**

Bases: *SuccessorsEngine*, *ProcedureMixin*

Execution engine based on Soot.

**lift_soot**(*addr=None*, *the_binary=None*, ***kwargs*)

**get_unconstrained_simprocedure**()

**classmethod setup_callsite**(*state*, *args*, *ret_addr*, *ret_var=None*)

**static setup_arguments**(*state*, *args*)

**static prepare_return_state**(*state*, *ret_value=None*)

**static terminate_execution**(*statement*, *state*, *successors*)

**static prepare_native_return_state**(*native_state*)

Hook target for native function call returns.

Recovers and stores the return value from native memory and toggles the state, s.t. execution continues in the Soot engine.

**class** angr.engines.**SuccessorsEngine**

Bases: *SimEngine*[*HeavyState*, *SimSuccessors*]

A mixin for SimEngine which implements `process` to perform common operations related to symbolic execution and dispatches to a `process_successors` method to fill a SimSuccessors object with the results.

**__init__**(*project*)

**Parameters**
**project** (Project)

**process**(*state*, ***kwargs*)

Perform execution with a state.

You should only override this method in a subclass in order to provide the correct method signature and docstring. You should override the _process method to do your actual execution.

**Parameters**

- **state** (*HeavyState*) – The state with which to execute. This state will be copied before modification.
- **inline** – This is an inline execution. Do not bother copying the state.
- **force_addr** – Force execution to pretend that we're working at this concrete address

**Return type**
*SimSuccessors*

**Returns**
A SimSuccessors object categorizing the execution's successor states

**process_successors**(*successors*, ***kwargs*)

Implement this function to fill out the SimSuccessors object with the results of stepping state.

In order to implement a model where multiple mixins can potentially handle a request, a mixin may implement this method and then perform a super() call if it wants to pass on handling to the next mixin.

Keep in mind python's method resolution order when composing multiple classes implementing this method. In short: left-to-right, depth-first, but deferring any base classes which are shared by multiple subclasses (the merge point of a diamond pattern in the inheritance graph) until the last point where they would be encountered in this depth-first search. For example, if you have classes A, B(A), C(B), D(A), E(C, D), then the method resolution order will be E, C, B, D, A.

**Parameters**

- **state** – The state to manipulate
- **successors** (*SimSuccessors*) – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

**class** angr.engines.**SuperFastpathMixin**

Bases: *VEXSlicingMixin*

This mixin implements the superfastpath execution mode, which skips all but the last four instructions.

**handle_vex_block**(*irsb*)

**class** angr.engines.**TrackActionsMixin**

Bases: *HeavyVEXMixin*

**__init__**(**args*, ***kwargs*)

**handle_vex_block**(*irsb*)

**class** angr.engines.**UberEngine**

Bases: *SimEngineFailure*, *SimEngineSyscall*, *HooksMixin*, *SimEngineUnicorn*, *SuperFastpathMixin*, *TrackActionsMixin*, *SimInspectMixin*, *HeavyResilienceMixin*, *SootMixin*, *AILMixin*, *HeavyVEXMixin*

The default execution engine for angr. This engine includes mixins for most common functionality in angr, including VEX IR, unicorn, syscall handling, and simprocedure handling.

For some performance-sensitive applications, you may want to create a custom engine with only the necessary mixins.

**class** angr.engines.**UberEnginePcode**

Bases: *SimEngineFailure*, *SimEngineSyscall*, *HooksMixin*, *HeavyPcodeMixin*

### Submodules

| | |
|---|---|
| *ail* | |
| *engine* | |
| *failure* | |
| *hook* | |
| *icicle* | icicle.py: An angr engine that uses Icicle to execute code. |
| *light* | |
| *pcode* | |
| *procedure* | |
| *soot* | |
| *successors* | |
| *syscall* | |
| *unicorn* | |
| *vex* | |

## 10.15.1 angr.engines.ail

**class** angr.engines.ail.**AILCallStack**

Bases: *CallStack*

An implementation of state.callstack for AIL symbolic execution

Contains extra state for local vars and AIL calling conventions

**__init__**(**args*, ***kwargs*)

**copy_without_tail**(*memo*)

**class** angr.engines.ail.**AILMixin**

Bases: *SuccessorsEngine*

The SuccessorsEngine mixin for providing AIL symbolic execution support

**class** angr.engines.ail.**SimEngineAILSimState**

Bases: *SimEngineLightAIL*[*StateType*, *DataType*, bool, None]

A light engine for symbolically executing AIL

**__init__**(*project*, *successors*)

**Parameters**

- **project** (Project)
- **successors** (SimSuccessors)

**process**(*state*, *, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
None

**Parameters**

- **state** (StateType)
- **block** (*Block | None*)
- **whitelist** (*set[int] | None*)

**property frame:** ***AILCallStack***

**lift_addr**(*addr*)

**Return type**
*Clinic*

**Parameters**
**addr** (*int*)

**lift**(*state*)

**Return type**
Block

**Parameters**
**state** (StateType | *int* | *Address*)

angr.engines.ail.**ail_call_state**(*project*, *start_addr*, *args*, *lifter*, *mode='symbolic'*, *options=None*, *add_options=None*, *remove_options=None*, *plugin_preset='default'*, *memory_cls=None*, *callstack_cls=None*)

**Parameters**

- **project** (Project)
- **start_addr** (*int | Address | str*)
- **args** (*Iterable[Bits]*)
- **lifter** (*Callable[[int], Clinic]*)
- **mode** (*str*)
- **options** (*set[str] | None*)
- **add_options** (*set[str] | None*)
- **remove_options** (*set[str] | None*)
- **plugin_preset** (*str*)
- **memory_cls** (type[MemoryMixin] | *None*)
- **callstack_cls** (type[AILCallStack] | *None*)

### Submodules


*callstack*
*engine_light*
*engine_successors*
*setup*


### angr.engines.ail.callstack

**class** angr.engines.ail.callstack.**AILCallStack**

Bases: *CallStack*

An implementation of state.callstack for AIL symbolic execution

Contains extra state for local vars and AIL calling conventions

**__init__**(**args*, ***kwargs*)

**copy_without_tail**(*memo*)

### angr.engines.ail.engine_light

**type** angr.engines.ail.engine_light.**StateType** = *SimState*[int, BV]

**type** angr.engines.ail.engine_light.**DataType** = Bits | Bool

**exception** angr.engines.ail.engine_light.**CallReached**

Bases: Exception

An exception to abort executing a block if we need to restart it with a call result

**class** angr.engines.ail.engine_light.**SimEngineAILSimState**

Bases: *SimEngineLightAIL*[*StateType*, *DataType*, bool, None]

A light engine for symbolically executing AIL

**__init__**(*project*, *successors*)

**Parameters**

- **project** (Project)
- **successors** (SimSuccessors)

**process**(*state*, *, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
None

**Parameters**

- **state** (StateType)
- **block** (*Block | None*)
- **whitelist** (*set[int] | None*)

**property frame:** *AILCallStack*

**lift_addr**(*addr*)

**Return type**
*Clinic*

**Parameters**
**addr** (*int*)

**lift**(*state*)

**Return type**
Block

**Parameters**
**state** (StateType | *int* | *Address*)

### angr.engines.ail.engine_successors

**class** angr.engines.ail.engine_successors.**AILMixin**

Bases: *SuccessorsEngine*

The SuccessorsEngine mixin for providing AIL symbolic execution support

### angr.engines.ail.setup

angr.engines.ail.setup.**ail_call_state**(*project*, *start_addr*, *args*, *lifter*, *mode='symbolic'*, *options=None*, *add_options=None*, *remove_options=None*, *plugin_preset='default'*, *memory_cls=None*, *callstack_cls=None*)

**Parameters**

- **project** (Project)
- **start_addr** (*int* | *Address* | *str*)
- **args** (*Iterable[Bits]*)
- **lifter** (*Callable[[int]*, Clinic])
- **mode** (*str*)
- **options** (*set[str]* | *None*)
- **add_options** (*set[str]* | *None*)
- **remove_options** (*set[str]* | *None*)
- **plugin_preset** (*str*)
- **memory_cls** (type[MemoryMixin] | *None*)
- **callstack_cls** (type[AILCallStack] | *None*)

## 10.15.2 angr.engines.engine

**class** angr.engines.engine.**SimEngine**

Bases: Generic

A SimEngine is a type which understands how to perform execution on a state.

**state:** *StateType*

**__init__**(*project*)

**Parameters**
**project** (Project)

### 10.15.3 angr.engines.failure

**class** angr.engines.failure.**SimEngineFailure**

Bases: *SuccessorsEngine*, *ProcedureMixin*

### 10.15.4 angr.engines.hook

**class** angr.engines.hook.**HooksMixin**

Bases: *SuccessorsEngine*, *ProcedureMixin*

A SimEngine mixin which adds a SimSuccessors handler which will look into the project's hooks and run the hook at the current address.

Will respond to the following parameters provided to the step stack:

- procedure: A SimProcedure instance to force-run instead of consulting the current hooks
- ret_to: An address to force-return-to at the end of the procedure

### 10.15.5 angr.engines.icicle

icicle.py: An angr engine that uses Icicle to execute code.

**class** angr.engines.icicle.**IcicleEngine**

Bases: *SuccessorsEngine*

An angr engine that uses Icicle to execute concrete states. The purpose of this implementation is to provide a high-performance concrete execution engine in angr. While historically, angr has focused on symbolic execution, better support for concrete execution enables new use cases such as fuzzing in angr. This is ideal for testing bespoke binary targets, such as microcontroller firmware, which may be difficult to correctly harness for use with traditional fuzzing engines.

This class is the base class for the Icicle engine. It implements execution by creating an Icicle instance, copying the state from angr to Icicle, and then running the Icicle instance. The results are then copied back to the angr state. The Icicle instance is cached on the engine and reused across runs: the first call takes a snapshot of the fresh VM, and subsequent calls either continue with the cached emu (same-run successors) or restore the snapshot and delta-sync the input state (branches).

For a more complete implementation, use the UberIcicleEngine class, which intends to provide a more complete set of features, such as hooks and syscalls.

**class** angr.engines.icicle.**UberIcicleEngine**

Bases: *SimEngineFailure*, *SimEngineSyscall*, *HooksMixin*, *IcicleEngine*

An extension of the IcicleEngine that uses mixins to add support for syscalls and hooks. Most users will prefer to use this engine instead of the IcicleEngine directly.

### 10.15.6 angr.engines.light

**class** angr.engines.light.**ArithmeticExpression**

Bases: object

**Add = 0**

Sub = 1

Or = 2

And = 4

RShift = 8

LShift = 16

Mul = 32

Xor = 64

CONST_TYPES = (<class 'int'>, <class 'angr.ailment.expression.Const'>)

__init__(*op*, *operands*)

op

operands

static try_unpack_const(*expr*)

class angr.engines.light.RegisterOffset

Bases: object

__init__(*bits*, *reg*, *offset*)

reg

offset

property bits

property symbolic

class angr.engines.light.SimEngineLight

Bases: *SimEngine*, Generic

A full-featured engine base class, suitable for static analysis

block: BlockType

state: *StateType*

stmt_idx: int

ins_addr: int

tmps: dict[int, *DataType_co*]

__init__(*project*, *logger=None*)

**Parameters**
**project** (Project)

**abstractmethod process**(*state*, *, *block=None*, ***kwargs*)

**Return type**
TypeVar(ResultType)

**Parameters**

- **state** (StateType)
- **block** (*BlockType | None*)

**lift**(*state*)

**Return type**
TypeVar(BlockType, bound= *BlockProtocol*)

**Parameters**
**state** (StateType)

**static sp_offset**(*bits*, *offset*)

**Return type**
BV

**Parameters**

- **bits** (*int*)
- **offset** (*int*)

**static extract_offset_to_sp**(*spoffset_expr*)

Extract the offset to the original stack pointer.

**Parameters**
**spoffset_expr** (Base) – The claripy AST to parse.

**Return type**
int | None

**Returns**
The offset to the original stack pointer, or None if *spoffset_expr* is not a supported type of SpOffset expression.

**class** angr.engines.light.**SimEngineLightAIL**

Bases: *SimEngineLight*[StateType, DataType_co, Block, ResultType], Generic

A mixin for doing static analysis on AIL

**__init__**(**args*, ***kwargs*)

**process**(*state*, *, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
TypeVar(ResultType)

**Parameters**

- **state** (StateType)
- **block** (*ailment.Block | None*)
- **whitelist** (*set[int] | None*)

class angr.engines.light.**SimEngineLightVEX**

Bases: *SimEngineLight*[StateType, DataType_co, *Block*, ResultType], Generic

A mixin for doing static analysis on VEX

**tyenv:** IRTypeEnv

**static unop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Unop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Unop], DataType_co]*)

**static binop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Binop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Binop], DataType_co]*)

**static binopv_handler**(*f*)

**Return type**
Callable[[TypeVar(T), int, int, Binop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, int, int, pyvex.expr.Binop], DataType_co]*)

**static triop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Triop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Triop], DataType_co]*)

**static qop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Qop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Qop], DataType_co]*)

**static ccall_handler**(*f*)

**Return type**
Callable[[TypeVar(T), CCall], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.CCall], DataType_co]*)

**static dirty_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Dirty], TypeVar(StmtDataType)]

**Parameters**
**f** (*Callable[[T, pyvex.stmt.Dirty], StmtDataType]*)

__init__(*args*, ***kwargs*)

process(*state*, ***, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
TypeVar(ResultType)

**Parameters**

- **state** (StateType)
- **block** (Block | *None*)
- **whitelist** (*set[int]* | *None*)

class angr.engines.light.**SimEngineNoexprAIL**

Bases: *SimEngineLightAIL*[StateType, DataType_co | None, StmtDataType, ResultType], Generic

A base class of SimEngineLightAIL that has default handlers for expressions if they just need to return None, so you don't have to implement every single expression handler as `return None`.

class angr.engines.light.**SimEngineNostmtAIL**

Bases: *SimEngineLightAIL*[StateType, DataType_co, StmtDataType | None, ResultType], Generic

A base class of SimEngineLightAIL that has default handlers for statements if they just need to return None, so you don't have to implement every single statement handler as `return None`.

class angr.engines.light.**SimEngineNostmtVEX**

Bases: *SimEngineLightVEX*[StateType, DataType_co, ResultType, None], Generic

A base class of SimEngineLightVEX that has default handlers for statements if they just need to return None, so you don't have to implement every single statement handler as `return None`.

class angr.engines.light.**SpOffset**

Bases: *RegisterOffset*

__init__(*bits*, *offset*, *is_base=False*)

**is_base**

## Submodules

*data*
*engine*

### angr.engines.light.data

class angr.engines.light.data.**ArithmeticExpression**

Bases: object

**Add = 0**

**Sub = 1**

**Or = 2**

**And = 4**

**RShift = 8**

LShift = 16

Mul = 32

Xor = 64

CONST_TYPES = (<class 'int'>, <class 'angr.ailment.expression.Const'>)

__init__(*op*, *operands*)

op

operands

static try_unpack_const(*expr*)

class angr.engines.light.data.RegisterOffset

Bases: object

__init__(*bits*, *reg*, *offset*)

reg

offset

property bits

property symbolic

class angr.engines.light.data.SpOffset

Bases: *RegisterOffset*

__init__(*bits*, *offset*, *is_base=False*)

is_base

### angr.engines.light.engine

class angr.engines.light.engine.BlockProtocol

Bases: Protocol

The minimum protocol that a block an engine can process should adhere to. Requires just an addr attribute.

addr: int

__init__(**args*, ***kwargs*)

class angr.engines.light.engine.IRTop

Bases: IRExpr

A dummy IRExpr used for intra-engine communication and code-reuse.

__init__(*ty*)

**Parameters**
**ty** (*str*)

result_type(*tyenv*)

class angr.engines.light.engine.**SimEngineLight**

Bases: *SimEngine*, Generic

A full-featured engine base class, suitable for static analysis

**block:** BlockType

**state:** *StateType*

**stmt_idx:** int

**ins_addr:** int

**tmps:** dict[int, *DataType_co*]

**__init__**(*project*, *logger=None*)

**Parameters**
**project** (Project)

**abstractmethod process**(*state*, *, *block=None*, **kwargs*)

**Return type**
TypeVar(ResultType)

**Parameters**

- **state** (StateType)
- **block** (*BlockType | None*)

**lift**(*state*)

**Return type**
TypeVar(BlockType, bound= *BlockProtocol*)

**Parameters**
**state** (StateType)

**static sp_offset**(*bits*, *offset*)

**Return type**
BV

**Parameters**

- **bits** (*int*)
- **offset** (*int*)

**static extract_offset_to_sp**(*spoffset_expr*)

Extract the offset to the original stack pointer.

**Parameters**
**spoffset_expr** (Base) – The claripy AST to parse.

**Return type**
int | None

**Returns**
The offset to the original stack pointer, or None if *spoffset_expr* is not a supported type of SpOffset expression.

angr.engines.light.engine.**longest_prefix_lookup**(*haystack*, *mapping*)

**Return type**
Optional[TypeVar(T)]

**Parameters**

- **haystack** (*str*)
- **mapping** (*dict[str, T]*)

**class** angr.engines.light.engine.**SimEngineLightVEX**

Bases: *SimEngineLight*[StateType, DataType_co, *Block*, ResultType], Generic

A mixin for doing static analysis on VEX

**tyenv:** IRTypeEnv

**static unop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Unop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Unop], DataType_co]*)

**static binop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Binop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Binop], DataType_co]*)

**static binopv_handler**(*f*)

**Return type**
Callable[[TypeVar(T), int, int, Binop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, int, int, pyvex.expr.Binop], DataType_co]*)

**static triop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Triop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Triop], DataType_co]*)

**static qop_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Qop], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.Qop], DataType_co]*)

**static ccall_handler**(*f*)

**Return type**
Callable[[TypeVar(T), CCall], TypeVar(DataType_co)]

**Parameters**
**f** (*Callable[[T, pyvex.expr.CCall], DataType_co]*)

**static dirty_handler**(*f*)

**Return type**
Callable[[TypeVar(T), Dirty], TypeVar(StmtDataType)]

**Parameters**
**f** (*Callable[[T, pyvex.stmt.Dirty], StmtDataType]*)

**__init__**(**args*, ***kwargs*)

**process**(*state*, ***, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
TypeVar(ResultType)

**Parameters**

- **state** (StateType)
- **block** (*Block | None*)
- **whitelist** (*set[int] | None*)

**class** angr.engines.light.engine.**SimEngineNostmtVEX**

Bases: *SimEngineLightVEX*[StateType, DataType_co, ResultType, None], Generic

A base class of SimEngineLightVEX that has default handlers for statements if they just need to return None, so you don't have to implement every single statement handler as `return None`.

**class** angr.engines.light.engine.**SimEngineLightAIL**

Bases: *SimEngineLight*[StateType, DataType_co, Block, ResultType], Generic

A mixin for doing static analysis on AIL

**__init__**(**args*, ***kwargs*)

**process**(*state*, ***, *block=None*, *whitelist=None*, ***kwargs*)

**Return type**
TypeVar(ResultType)

**Parameters**

- **state** (StateType)
- **block** (*ailment.Block | None*)
- **whitelist** (*set[int] | None*)

**class** angr.engines.light.engine.**SimEngineNostmtAIL**

Bases: *SimEngineLightAIL*[StateType, DataType_co, StmtDataType | None, ResultType], Generic

A base class of SimEngineLightAIL that has default handlers for statements if they just need to return None, so you don't have to implement every single statement handler as `return None`.

**class** angr.engines.light.engine.**SimEngineNoexprAIL**

Bases: *SimEngineLightAIL*[StateType, DataType_co | None, StmtDataType, ResultType], Generic

A base class of SimEngineLightAIL that has default handlers for expressions if they just need to return None, so you don't have to implement every single expression handler as `return None`.

### 10.15.7 angr.engines.pcode

**class** angr.engines.pcode.**HeavyPcodeMixin**

Bases: *SuccessorsEngine*, *PcodeLifterEngineMixin*, *PcodeEmulatorMixin*

Execution engine based on P-code, Ghidra's IR.

Responds to the following parameters to the step stack:

- irsb: The P-Code IRSB object to use for execution. If not provided one will be lifted.
- skip_stmts: The number of statements to skip in processing
- last_stmt: Do not execute any statements after this statement
- thumb: Whether the block should be force to be lifted in ARM's THUMB mode. (FIXME)
- **extra_stop_points:**
  An extra set of points at which to break basic blocks
- insn_bytes: A string of bytes to use for the block instead of the project.
- size: The maximum size of the block, in bytes.
- num_inst: The maximum number of instructions.

**__init__**(**args*, ***kwargs*)

angr.engines.pcode.**register_pcode_arch_default_cc**(*arch*)

**Parameters**
**arch** (*ArchPcode*)

#### Submodules

| |
|---|
| *behavior* |
| *cc* |
| *emulate* |
| *engine* |
| *lifter* |

#### angr.engines.pcode.behavior

angr.engines.pcode.behavior.**make_bv_sizes_equal**(*bv1*, *bv2*, *zero_ext=False*)

Makes two BVs equal in length through sign extension.

**Return type**
tuple[BV, BV]

**Parameters**

- **bv1** (*BV*)
- **bv2** (*BV*)
- **zero_ext** (*bool*)

**class** angr.engines.pcode.behavior.**OpBehavior**

Bases: object

Base class for all operation behaviors.

**__init__**(*opcode*, *is_unary*, *is_special=False*)

**Parameters**

- **opcode** (*OpCode*)
- **is_unary** (*bool*)
- **is_special** (*bool*)

**Return type**
None

**opcode:** OpCode

**is_unary:** bool

**is_special:** bool

**evaluate_unary**(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**static generic_compare**(*args*, *comparison*)

**Return type**
BV

**Parameters**

- **args** (*Iterable[BV]*)
- **comparison** (*Callable[[BV, BV], BV]*)

**classmethod booleanize**(*in1*)

Reduce input BV to a single bit of truth: out <- 1 if (in1 != 0) else 0.

**Return type**
BV

**Parameters**
**in1** (*BV*)

class angr.engines.pcode.behavior.**OpBehaviorCopy**

Bases: *OpBehavior*

Behavior for the COPY operation.

**__init__**()

**evaluate_unary**(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorEqual**

Bases: *OpBehavior*

Behavior for the INT_EQUAL operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorNotEqual**

Bases: *OpBehavior*

Behavior for the INT_NOTEQUAL operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorIntSless**

Bases: *OpBehavior*

Behavior for the INT_SLESS operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorIntSlessEqual**

Bases: *OpBehavior*

Behavior for the INT_SLESSEQUAL operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)

- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntLess**

Bases: *OpBehavior*

Behavior for the INT_LESS operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntLessEqual**

Bases: *OpBehavior*

Behavior for the INT_LESSEQUAL operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntZext**

Bases: *OpBehavior*

Behavior for the INT_ZEXT operation.

**__init__**()

**evaluate_unary**(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntSext**

Bases: *OpBehavior*

Behavior for the INT_SEXT operation.

**__init__**()

**evaluate_unary**(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntAdd**

Bases: *OpBehavior*

Behavior for the INT_ADD operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntSub**

Bases: *OpBehavior*

Behavior for the INT_SUB operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntCarry**

Bases: *OpBehavior*

Behavior for the INT_CARRY operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntScarry**

Bases: *OpBehavior*

Behavior for the INT_SCARRY operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntSborrow**

Bases: *OpBehavior*

Behavior for the INT_SBORROW operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorInt2Comp**

Bases: *OpBehavior*

Behavior for the INT_2COMP operation.

__init__()

evaluate_unary(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

is_special: bool

is_unary: bool

opcode: OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntNegate**

Bases: *OpBehavior*

Behavior for the INT_NEGATE operation.

__init__()

evaluate_unary(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

is_special: bool

is_unary: bool

opcode: OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntXor**

Bases: *OpBehavior*

Behavior for the INT_XOR operation.

__init__()

evaluate_binary(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorIntAnd**

Bases: *OpBehavior*

Behavior for the INT_AND operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorIntOr**

Bases: *OpBehavior*

Behavior for the INT_OR operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntLeft**

Bases: *OpBehavior*

Behavior for the INT_LEFT operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntRight**

Bases: *OpBehavior*

Behavior for the INT_RIGHT operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntSright**

Bases: *OpBehavior*

Behavior for the INT_SRIGHT operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorIntMult**

Bases: *OpBehavior*

Behavior for the INT_MULT operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorIntDiv**

Bases: *OpBehavior*

Behavior for the INT_DIV operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)

- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntSdiv**

Bases: *OpBehavior*

Behavior for the INT_SDIV operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntRem**

Bases: *OpBehavior*

Behavior for the INT_REM operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorIntSrem**

Bases: *OpBehavior*

Behavior for the INT_SREM operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorBoolNegate**

Bases: *OpBehavior*

Behavior for the BOOL_NEGATE operation.

**__init__**()

**evaluate_unary**(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorBoolXor**

Bases: *OpBehavior*

Behavior for the BOOL_XOR operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorBoolAnd**

Bases: *OpBehavior*

Behavior for the BOOL_AND operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorBoolOr**

Bases: *OpBehavior*

Behavior for the BOOL_OR operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)

- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatEqual**

Bases: *OpBehavior*

Behavior for the FLOAT_EQUAL operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatNotEqual**

Bases: *OpBehavior*

Behavior for the FLOAT_NOTEQUAL operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatLess**

Bases: *OpBehavior*

Behavior for the FLOAT_LESS operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatLessEqual**

Bases: *OpBehavior*

Behavior for the FLOAT_LESSEQUAL operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatNan**

Bases: *OpBehavior*

Behavior for the FLOAT_NAN operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatAdd**

Bases: *OpBehavior*

Behavior for the FLOAT_ADD operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatDiv**

Bases: *OpBehavior*

Behavior for the FLOAT_DIV operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatMult**

Bases: *OpBehavior*

Behavior for the FLOAT_MULT operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatSub**

Bases: *OpBehavior*

Behavior for the FLOAT_SUB operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatNeg**

Bases: *OpBehavior*

Behavior for the FLOAT_NEG operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatAbs**

Bases: *OpBehavior*

Behavior for the FLOAT_ABS operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatSqrt**

Bases: *OpBehavior*

Behavior for the FLOAT_SQRT operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatInt2Float**

Bases: *OpBehavior*

Behavior for the FLOAT_INT2FLOAT operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorFloatFloat2Float**

Bases: *OpBehavior*

Behavior for the FLOAT_FLOAT2FLOAT operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatTrunc**

Bases: *OpBehavior*

Behavior for the FLOAT_TRUNC operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatCeil**

Bases: *OpBehavior*

Behavior for the FLOAT_CEIL operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatFloor**

Bases: *OpBehavior*

Behavior for the FLOAT_FLOOR operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorFloatRound**

Bases: *OpBehavior*

Behavior for the FLOAT_ROUND operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

class angr.engines.pcode.behavior.**OpBehaviorPiece**

Bases: *OpBehavior*

Behavior for the PIECE operation.

**__init__**()

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorSubpiece**

Bases: *OpBehavior*

Behavior for the SUBPIECE operation.

**__init__**()

**evaluate_binary**(*size_out*, *size_in*, *in1*, *in2*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)
- **in2** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorPopcount**

Bases: *OpBehavior*

Behavior for the POPCOUNT operation.

**__init__**()

**evaluate_unary**(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

**is_special:** bool

**is_unary:** bool

**opcode:** OpCode

**class** angr.engines.pcode.behavior.**OpBehaviorLzcount**

Bases: *OpBehavior*

Behavior for the LZCOUNT operation.

__init__()

evaluate_unary(*size_out*, *size_in*, *in1*)

**Return type**
BV

**Parameters**

- **size_out** (*int*)
- **size_in** (*int*)
- **in1** (*BV*)

is_special: bool

is_unary: bool

opcode: OpCode

class angr.engines.pcode.behavior.**BehaviorFactory**

Bases: object

Returns the behavior object for a given opcode.

__init__()

get_behavior_for_opcode(*opcode*)

**Return type**
*OpBehavior*

**Parameters**
**opcode** (*OpCode*)

### angr.engines.pcode.cc

class angr.engines.pcode.cc.**SimCCPCodeBase**

Bases: *SimCC*

Base class for all pcode calling conventions.

LANGUAGE = None

classmethod ARCH()

class angr.engines.pcode.cc.**SimCCM68k**

Bases: *SimCCPCodeBase*

Default CC for M68k

LANGUAGE = '68000:BE:32:default'

ARG_REGS: list[str] = []

FP_ARG_REGS: list[str] = []

STACKARG_SP_DIFF = 4

RETURN_VAL: *SimFunctionArgument* | None = <d0>

RETURN_ADDR: *SimFunctionArgument* | None = [0x0]

class angr.engines.pcode.cc.SimCCRISCV

Bases: *SimCCPCodeBase*

Default CC for RISCV

LANGUAGE = 'RISCV:LE:32:RV32G'

ARG_REGS: list[str] = ['a0', 'a1', 'a2', 'a3', 'a4', 'a5', 'a6', 'a7']

RETURN_ADDR: *SimFunctionArgument* | None = <ra>

RETURN_VAL: *SimFunctionArgument* | None = <a0>

class angr.engines.pcode.cc.SimCCSPARC

Bases: *SimCCPCodeBase*

Default CC for SPARC

LANGUAGE = 'sparc:BE:32:default'

ARG_REGS: list[str] = ['o0', 'o1', 'o2', 'o3', 'o4', 'o5']

RETURN_VAL: *SimFunctionArgument* | None = <o0>

RETURN_ADDR: *SimFunctionArgument* | None = <o7>

class angr.engines.pcode.cc.SimCCSH4

Bases: *SimCCPCodeBase*

Default CC for SH4

LANGUAGE = 'SuperH4:LE:32:default'

ARG_REGS: list[str] = ['r4', 'r5']

RETURN_VAL: *SimFunctionArgument* | None = <r0>

RETURN_ADDR: *SimFunctionArgument* | None = <pr>

class angr.engines.pcode.cc.SimCCPARISC

Bases: *SimCCPCodeBase*

Default CC for PARISC

LANGUAGE = 'pa-risc:BE:32:default'

ARG_REGS: list[str] = ['r26', 'r25']

RETURN_VAL: *SimFunctionArgument* | None = <r28>

RETURN_ADDR: *SimFunctionArgument* | None = <rp>

class angr.engines.pcode.cc.SimCCPowerPC

Bases: *SimCCPCodeBase*

Default CC for PowerPC

LANGUAGE = 'PowerPC:BE:32:e200'

ARG_REGS: list[str] = ['r3', 'r4', 'r5', 'r6', 'r7', 'r8', 'r9', 'r10']

FP_ARG_REGS: list[str] = []

STACKARG_SP_BUFF = 8

RETURN_ADDR: *SimFunctionArgument* | None = <lr>

RETURN_VAL: *SimFunctionArgument* | None = <r3>

class angr.engines.pcode.cc.**SimCCXtensa**

Bases: *SimCCPCodeBase*

Default CC for Xtensa

LANGUAGE = 'Xtensa:LE:32:default'

ARG_REGS: list[str] = ['a2', 'a3', 'a4', 'a5', 'a6', 'a7']

FP_ARG_REGS: list[str] = []

RETURN_ADDR: *SimFunctionArgument* | None = <a0>

RETURN_VAL: *SimFunctionArgument* | None = <a2>

angr.engines.pcode.cc.**register_pcode_arch_default_cc**(*arch*)

**Parameters**
**arch** (*ArchPcode*)

### angr.engines.pcode.emulate

class angr.engines.pcode.emulate.**PcodeEmulatorMixin**

Bases: *SimEngine*

Mixin for p-code execution.

**__init__**(**args*, ***kwargs*)

**handle_pcode_block**(*irsb*)

Execute a single P-Code IRSB.

**Parameters**
**irsb** (*IRSB*) – Block to be executed.

**Return type**
None

### angr.engines.pcode.engine

class angr.engines.pcode.engine.**HeavyPcodeMixin**

Bases: *SuccessorsEngine*, *PcodeLifterEngineMixin*, *PcodeEmulatorMixin*

Execution engine based on P-code, Ghidra's IR.

Responds to the following parameters to the step stack:

- irsb: The P-Code IRSB object to use for execution. If not provided one will be lifted.
- skip_stmts: The number of statements to skip in processing
- last_stmt: Do not execute any statements after this statement
- thumb: Whether the block should be force to be lifted in ARM's THUMB mode. (FIXME)

- **extra_stop_points:**
  An extra set of points at which to break basic blocks
- insn_bytes: A string of bytes to use for the block instead of the project.
- size: The maximum size of the block, in bytes.
- num_inst: The maximum number of instructions.

**__init__**(**args*, ***kwargs*)

### angr.engines.pcode.lifter

**class** angr.engines.pcode.lifter.**ExitStatement**

Bases: object

This class exists to ease compatibility with CFGFast's processing of exit_statements. See _scan_irsb method.

**__init__**(*dst*, *jumpkind*)

**Parameters**

- **dst** (*int | None*)
- **jumpkind** (*str*)

**dst:** int | None

**jumpkind:** str

**class** angr.engines.pcode.lifter.**PcodeDisassemblerBlock**

Bases: *DisassemblerBlock*

Helper class to represent a block of disassembled target architecture instructions

**addr**

**arch**

**insns**

**thumb**

**class** angr.engines.pcode.lifter.**PcodeDisassemblerInsn**

Bases: *DisassemblerInsn*

Helper class to represent a disassembled target architecture instruction

**__init__**(*pcode_insn*)

**property size:** int

**property address:** int

**property mnemonic:** str

**property op_str:** str

**class** angr.engines.pcode.lifter.**IRSB**

Bases: object

IRSB stands for *Intermediate Representation Super-Block*. An IRSB in is a single-entry, multiple-exit code block.

**Variables**

- **arch** (archinfo.Arch) – The architecture this block is lifted under
- **statements** (list of IRStmt) – The statements in this block
- **next** (IRExpr) – The expression for the default exit target of this block
- **offsIP** (*int*) – The offset of the instruction pointer in the VEX guest state
- **stmts_used** (*int*) – The number of statements in this IRSB
- **jumpkind** (*str*) – The type of this block's default jump (call, boring, syscall, etc) as a VEX enum string
- **direct_next** (*bool*) – Whether this block ends with a direct (not indirect) jump or branch
- **size** (*int*) – The size of this block in bytes
- **addr** (*int*) – The address of this basic block, i.e. the address in the first IMark

**MAX_EXITS = 400**

**MAX_DATA_REFS = 2000**

**__init__**(*data*, *mem_addr*, *arch*, *max_inst=None*, *max_bytes=None*, *bytes_offset=0*, *traceflags=0*, *opt_level=1*, *num_inst=None*, *num_bytes=None*, *strict_block_end=False*, *skip_stmts=False*, *collect_data_refs=False*)

**Parameters**

- **data** (str | bytes | None) – The bytes to lift. Can be either a string of bytes or a cffi buffer object. You may also pass None to initialize an empty IRSB.
- **mem_addr** (int) – The address to lift the data at.
- **arch** (Arch) – The architecture to lift the data as.
- **max_inst** (int | None) – The maximum number of instructions to lift. (See note below)
- **max_bytes** (int | None) – The maximum number of bytes to use.
- **num_inst** (int | None) – Replaces max_inst if max_inst is None. If set to None as well, no instruction limit is used.
- **num_bytes** (int | None) – Replaces max_bytes if max_bytes is None. If set to None as well, no byte limit is used.
- **bytes_offset** (int) – The offset into *data* to start lifting at. Note that for ARM THUMB mode, both *mem_addr* and *bytes_offset* must be odd (typically *bytes_offset* is set to 1).
- **traceflags** (int) – Unused by P-Code lifter
- **opt_level** (int) – Unused by P-Code lifter
- **strict_block_end** (bool) – Unused by P-Code lifter
- **skip_stmts** (*bool*)
- **collect_data_refs** (*bool*)

**Return type**
None

> **Note**
>
> Explicitly specifying the number of instructions to lift (*max_inst*) may not always work exactly as expected. For example, on MIPS, it is meaningless to lift a branch or jump instruction without its delay slot. VEX attempts to Do The Right Thing by possibly decoding fewer instructions than requested. Specifically, this means that lifting a branch or jump on MIPS as a single instruction (*max_inst=1*) will result in an empty IRSB, and subsequent attempts to run this block will raise *SimIRSBError('Empty IRSB passed to SimIRSB.')*.

> **Note**
>
> If no instruction and byte limit is used, the lifter will continue lifting the block until the block ends properly or until it runs out of data to lift.

**addr:** int

**arch:** Arch

**behaviors:** *BehaviorFactory* | None

**data_refs:** Sequence

**const_vals:** Sequence

**default_exit_target:** Any

**jumpkind:** str | None

**next:** IRExpr | None

**static empty_block**(*arch*, *addr*, *statements=None*, *nxt=None*, *tyenv=None*, *jumpkind=None*, *direct_next=None*, *size=None*)

**Return type**
*IRSB*

**Parameters**

- **arch** (*Arch*)
- **addr** (*int*)
- **statements** (*Sequence | None*)
- **nxt** (*int | None*)
- **jumpkind** (*str | None*)
- **direct_next** (*bool | None*)
- **size** (*int | None*)

**property has_statements:** bool

**property exit_statements:** Sequence[tuple[int, int, *ExitStatement*]]

**copy**()

Copy by creating an empty IRSB and then filling in the leftover attributes. Copy is made as deep as possible

**Return type**
*IRSB*

**extend**(*extendwith*)

Appends an irsb to the current irsb. The irsb that is appended is invalidated. The appended irsb's jumpkind and default exit are used. :type extendwith: *IRSB* :param extendwith: The IRSB to append to this IRSB

**Return type**
*IRSB*

**Parameters**
**extendwith** (IRSB)

**invalidate_direct_next**()

**Return type**
None

**pp**()

Pretty-print the IRSB to stdout.

**Return type**
None

**property tyenv**

**property stmts_used: int**

**property offsIP: int | None**

**property direct_next: bool**

**property expressions**

Return an iterator of all expressions contained in the IRSB.

**property instructions: int**

The number of instructions in this block

**property instruction_addresses: Sequence[int]**

Addresses of instructions in this block.

**property size: int**

The size of this block, in bytes

**property operations**

A list of all operations done by the IRSB, as libVEX enum names

**property all_constants**

Returns all constants in the block (including incrementing of the program counter) as pyvex.const.IRConst.

**property constants**

The constants (excluding updates of the program counter) in the IRSB as pyvex.const.IRConst.

**property constant_jump_targets**

A set of the static jump targets of the basic block.

**property constant_jump_targets_and_jumpkinds**

A dict of the static jump targets of the basic block to their jumpkind.

**property is_noop_block: bool**

Returns True if this block is a no-op block (i.e. it has no instructions and no jumps).

**property statements: list**

**property disassembly: *PcodeDisassemblerBlock***

**class** angr.engines.pcode.lifter.**Lifter**

Bases: object

A lifter is a class of methods for processing a block.

**Variables**

- **data** – The bytes to lift as either a python string of bytes or a cffi buffer object.
- **bytes_offset** – The offset into *data* to start lifting at.
- **max_bytes** – The maximum number of bytes to lift. If set to None, no byte limit is used.
- **max_inst** – The maximum number of instructions to lift. If set to None, no instruction limit is used.
- **opt_level** – Unused by P-Code lifter
- **traceflags** – Unused by P-Code lifter
- **allow_arch_optimizations** – Unused by P-Code lifter
- **strict_block_end** – Unused by P-Code lifter
- **skip_stmts** – Unused by P-Code lifter

**REQUIRE_DATA_C = False**

**REQUIRE_DATA_PY = False**

**__init__**(*arch*, *addr*)

**Parameters**

- **arch** (*Arch*)
- **addr** (*int*)

**arch: Arch**

**addr: int**

**data: str | *bytes* | None**

**bytes_offset: int | None**

**opt_level: int**

**traceflags: int | None**

**allow_arch_optimizations: bool | None**

**strict_block_end: bool | None**

**collect_data_refs:** bool

**max_inst:** int | None

**max_bytes:** int | None

**skip_stmts:** bool

**irsb:** *IRSB*

**lift**()

Lifts the data using the information passed into _lift. Should be overridden in child classes.

Should set the lifted IRSB to self.irsb. If a lifter raises a LiftingException on the data, this signals that the lifter cannot lift this data and arch and the lifter is skipped. If a lifter can lift any amount of data, it should lift it and return the lifted block with a jumpkind of Ijk_NoDecode, signalling to pyvex that other lifters should be used on the undecodable data.

**Return type**
None

angr.engines.pcode.lifter.**lift**(*data*, *addr*, *arch*, *max_bytes=None*, *max_inst=None*, *bytes_offset=0*, *opt_level=1*, *traceflags=0*, *strict_block_end=True*, *inner=False*, *skip_stmts=False*, *collect_data_refs=False*, *block_lifter=None*)

Lift machine code in *data* to a P-code IRSB.

If a lifter raises a LiftingException on the data, it is skipped. If it succeeds and returns a block with a jumpkind of Ijk_NoDecode, all of the lifters are tried on the rest of the data and if they work, their output is appended to the first block.

**Parameters**

- **arch** (Arch) – The arch to lift the data as.
- **addr** (int) – The starting address of the block. Effects the IMarks.
- **data** (str | bytes | None) – The bytes to lift as either a python string of bytes or a cffi buffer object.
- **max_bytes** (int | None) – The maximum number of bytes to lift. If set to None, no byte limit is used.
- **max_inst** (int | None) – The maximum number of instructions to lift. If set to None, no instruction limit is used.
- **bytes_offset** (int) – The offset into *data* to start lifting at.
- **opt_level** (int) – Unused by P-Code lifter
- **traceflags** (int) – Unused by P-Code lifter
- **block_lifter** (*PcodeBasicBlockLifter* | None) – The basic block lifter, and therefore the Sleigh context, to decode with. A private one is created when it is not given.
- **strict_block_end** (*bool*)
- **inner** (*bool*)
- **skip_stmts** (*bool*)
- **collect_data_refs** (*bool*)

**Return type**
*IRSB*

> **Note**
>
> Explicitly specifying the number of instructions to lift (*max_inst*) may not always work exactly as expected. For example, on MIPS, it is meaningless to lift a branch or jump instruction without its delay slot. VEX attempts to Do The Right Thing by possibly decoding fewer instructions than requested. Specifically, this means that lifting a branch or jump on MIPS as a single instruction (*max_inst=1*) will result in an empty IRSB, and subsequent attempts to run this block will raise *SimIRSBError('Empty IRSB passed to SimIRSB.')*.

> **Note**
>
> If no instruction and byte limit is used, the lifter will continue lifting the block until the block ends properly or until it runs out of data to lift.

**class** angr.engines.pcode.lifter.**PcodeBasicBlockLifter**

Bases: object

Lifts basic blocks to P-code

Sleigh records context variables per address, so an instance must stay with one binary. See *angr.project.Project.pcode_block_lifter()*.

**__init__**(*arch*)

**Parameters**
**arch** (*Arch*)

**arch:** Arch

**context:** Context

**behaviors:** *BehaviorFactory*

**lift**(*irsb*, *baseaddr*, *data*, *bytes_offset=0*, *max_bytes=None*, *max_inst=None*, *branch_delay_slot=False*, *is_sparc32=False*)

**Return type**
None

**Parameters**

- **irsb** (IRSB)
- **baseaddr** (*int*)
- **data** (bytes | *bytearray*)
- **bytes_offset** (*int*)
- **max_bytes** (*int* | *None*)
- **max_inst** (*int* | *None*)
- **branch_delay_slot** (*bool*)
- **is_sparc32** (*bool*)

**class** angr.engines.pcode.lifter.**PcodeLifter**

Bases: *Lifter*

Handles calling into pypcode to lift a block

__init__(*arch*, *addr*, *block_lifter*)

**Parameters**

- **arch** (*Arch*)
- **addr** (*int*)
- **block_lifter** (PcodeBasicBlockLifter)

**block_lifter:** *PcodeBasicBlockLifter*

class angr.engines.pcode.lifter.**PcodeLifterEngineMixin**

Bases: *SimEngine*

Lifter mixin to lift from machine code to P-Code.

__init__(*project=None*, *use_cache=None*, *cache_size=50000*, *default_opt_level=1*, *selfmodifying_code=None*, *single_step=False*, *default_strict_block_end=False*, ***kwargs*)

**Parameters**

- **use_cache** (*bool | None*)
- **cache_size** (*int*)
- **default_opt_level** (*int*)
- **selfmodifying_code** (*bool | None*)
- **single_step** (*bool*)
- **default_strict_block_end** (*bool*)

clear_cache()

**Return type**
None

lift_vex(*addr=None*, *state=None*, *clemory=None*, *insn_bytes=None*, *arch=None*, *size=None*, *num_inst=None*, *traceflags=0*, *thumb=False*, *extra_stop_points=None*, *opt_level=None*, *strict_block_end=None*, *skip_stmts=False*, *collect_data_refs=False*, *load_from_ro_regions=False*, *cross_insn_opt=None*, *const_prop=None*)

Temporary compatibility interface for integration with block code.

**Return type**
*IRSB*

**Parameters**

- **addr** (*int | None*)
- **state** (SimState | *None*)
- **clemory** (*Clemory | ClemoryReadOnlyView | None*)
- **insn_bytes** (bytes | *None*)
- **arch** (*Arch | None*)
- **size** (*int | None*)
- **num_inst** (*int | None*)
- **traceflags** (*int*)
- **thumb** (*bool*)

- **extra_stop_points** (*Iterable[int] | None*)
- **opt_level** (*int | None*)
- **strict_block_end** (*bool | None*)
- **skip_stmts** (*bool*)
- **collect_data_refs** (*bool*)
- **load_from_ro_regions** (*bool*)
- **cross_insn_opt** (*bool | None*)
- **const_prop** (*bool | None*)

**lift_pcode**(*addr=None*, *state=None*, *clemory=None*, *insn_bytes=None*, *arch=None*, *size=None*, *num_inst=None*, *traceflags=0*, *thumb=False*, *extra_stop_points=None*, *opt_level=None*, *strict_block_end=None*, *skip_stmts=False*, *collect_data_refs=False*, *load_from_ro_regions=False*, *cross_insn_opt=None*, *const_prop=None*)

Lift an IRSB.

There are many possible valid sets of parameters. You at the very least must pass some source of data, some source of an architecture, and some source of an address.

Sources of data in order of priority: insn_bytes, clemory, state

Sources of an address, in order of priority: addr, state

Sources of an architecture, in order of priority: arch, clemory, state

**Parameters**

- **state** (*SimState* | None) – A state to use as a data source.
- **clemory** (Clemory | ClemoryReadOnlyView | None) – A cle.memory.Clemory object to use as a data source.
- **addr** (int | None) – The address at which to start the block.
- **thumb** (bool) – Whether the block should be lifted in ARM's THUMB mode.
- **opt_level** (int | None) – Unused for P-Code lifter
- **insn_bytes** (bytes | None) – A string of bytes to use as a data source.
- **size** (int | None) – The maximum size of the block, in bytes.
- **num_inst** (int | None) – The maximum number of instructions.
- **traceflags** (int) – Unused by P-Code lifter
- **strict_block_end** (bool | None) – Unused by P-Code lifter
- **load_from_ro_regions** (bool) – Unused by P-Code lifter
- **arch** (*Arch | None*)
- **extra_stop_points** (*Iterable[int] | None*)
- **skip_stmts** (*bool*)
- **collect_data_refs** (*bool*)
- **cross_insn_opt** (*bool | None*)
- **const_prop** (*bool | None*)

**Return type**
*IRSB*

### 10.15.8 angr.engines.procedure

**class** angr.engines.procedure.**ProcedureMixin**

Bases: object

A mixin for SimEngine which adds the process_procedure method for calling a SimProcedure and adding its results to a SimSuccessors.

**process_procedure**(*state*, *successors*, *procedure*, *ret_to=None*, *arguments=None*, ***kwargs*)

**class** angr.engines.procedure.**ProcedureEngine**

Bases: *ProcedureMixin*, *SuccessorsEngine*

A SimEngine that you may use if you only care about processing SimProcedures. *Requires* the procedure kwarg to be passed to process.

### 10.15.9 angr.engines.soot

**class** angr.engines.soot.**SootMixin**

Bases: *SuccessorsEngine*, *ProcedureMixin*

Execution engine based on Soot.

**lift_soot**(*addr=None*, *the_binary=None*, ***kwargs*)

**get_unconstrained_simprocedure**()

**classmethod setup_callsite**(*state*, *args*, *ret_addr*, *ret_var=None*)

**static setup_arguments**(*state*, *args*)

**static prepare_return_state**(*state*, *ret_value=None*)

**static terminate_execution**(*statement*, *state*, *successors*)

**static prepare_native_return_state**(*native_state*)

Hook target for native function call returns.

Recovers and stores the return value from native memory and toggles the state, s.t. execution continues in the Soot engine.

#### Submodules

*engine*
*exceptions*
*expressions*
*field_dispatcher*
*method_dispatcher*
*statements*
*values*

### angr.engines.soot.engine

**class** angr.engines.soot.engine.**SootMixin**

Bases: *SuccessorsEngine*, *ProcedureMixin*

Execution engine based on Soot.

**lift_soot**(*addr=None*, *the_binary=None*, ***kwargs*)

**get_unconstrained_simprocedure**()

**classmethod setup_callsite**(*state*, *args*, *ret_addr*, *ret_var=None*)

**static setup_arguments**(*state*, *args*)

**static prepare_return_state**(*state*, *ret_value=None*)

**static terminate_execution**(*statement*, *state*, *successors*)

**static prepare_native_return_state**(*native_state*)

Hook target for native function call returns.

Recovers and stores the return value from native memory and toggles the state, s.t. execution continues in the Soot engine.

### angr.engines.soot.exceptions

**exception** angr.engines.soot.exceptions.**BlockTerminationNotice**

Bases: Exception

**exception** angr.engines.soot.exceptions.**IncorrectLocationException**

Bases: Exception

**exception** angr.engines.soot.exceptions.**SootMethodNotLoadedException**

Bases: Exception

**exception** angr.engines.soot.exceptions.**SootFieldNotLoadedException**

Bases: Exception

### angr.engines.soot.expressions

**class** angr.engines.soot.expressions.**SimSootExpr_ArrayRef**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.**SimSootExpr_Binop**

Bases: *SimSootExpr*

**binop_str_to_function = {'add': <built-in function add>, 'and': <built-in function and_>, 'cmp': <built-in function ne>, 'cmpl': <built-in function lt>, 'div': <built-in function truediv>, 'mul': <built-in function mul>, 'or': <built-in function or_>, 'rem': <built-in function imod>, 'shl': <built-in function lshift>, 'shr': <built-in function rshift>, 'sub': <built-in function sub>, 'xor': <built-in function xor>}**

**class** angr.engines.soot.expressions.**SimSootExpr_Cast**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.**SimSootExpr_ClassConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_Condition**

Bases: *SimSootExpr*

**condition_str_to_function = {'eq': <built-in function eq>, 'ge': <built-in function SGE>, 'gt': <built-in function SGT>, 'le': <built-in function SLE>, 'lt': <built-in function SLT>, 'ne': <built-in function ne>}**

class angr.engines.soot.expressions.**SimSootExpr_DoubleConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_FloatConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_InstanceFieldRef**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_InstanceOf**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_IntConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_InterfaceInvoke**

Bases: *SimSootExpr_VirtualInvoke*

class angr.engines.soot.expressions.**SimSootExpr_Length**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_Local**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_LongConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_New**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_NewArray**

Bases: *SimSootExpr*

**static new_array**(*state*, *element_type*, *size*)

Allocates a new array in memory and returns the reference to the base.

class angr.engines.soot.expressions.**SimSootExpr_NewMultiArray**

Bases: *SimSootExpr*

**static new_array**(*state*, *element_type*, *size*, *default_value_generator=None*)

Allocates a new multi array in memory and returns the reference to the base.

class angr.engines.soot.expressions.**SimSootExpr_NullConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_ParamRef**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.**SimSootExpr_Phi**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.**SimSootExpr_SpecialInvoke**

Bases: *InvokeBase*

Special invocations are used either for invoking instance methods of a superclass or a constructor. Compared to virtual invokes, the class containing the method is passed explicitly in the invoke expression (@expr.class_name) rather than determined dynamically by the type of the base objects.

**class** angr.engines.soot.expressions.**SimSootExpr_StaticFieldRef**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.**SimSootExpr_StaticInvoke**

Bases: *InvokeBase*

**class** angr.engines.soot.expressions.**SimSootExpr_StringConstant**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.**SimSootExpr_ThisRef**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.**SimSootExpr_Unsupported**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.**SimSootExpr_VirtualInvoke**

Bases: *InvokeBase*

Instance methods are dynamically resolved by the actual type of the base object (i.e. the object on which the method is invoked) and *not* by the type of the variable storing the reference.

angr.engines.soot.expressions.**translate_expr**(*expr*, *state*)

### Submodules

| |
|---|
| *arrayref* |
| *base* |
| *binop* |
| *cast* |
| *condition* |
| *constants* |
| *instanceOf* |
| *instancefieldref* |
| *invoke* |
| *length* |
| *local* |
| *new* |
| *newArray* |
| *newMultiArray* |
| *paramref* |
| *phi* |
| *staticfieldref* |
| *thisref* |
| *unsupported* |

### angr.engines.soot.expressions.arrayref

class angr.engines.soot.expressions.arrayref.**SimSootExpr_ArrayRef**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.base

class angr.engines.soot.expressions.base.**SimSootExpr**

Bases: object

**__init__**(*expr*, *state*)

**process**()

### angr.engines.soot.expressions.binop

class angr.engines.soot.expressions.binop.**SimSootExpr_Binop**

Bases: *SimSootExpr*

**binop_str_to_function = {'add': <built-in function add>, 'and': <built-in function and_>, 'cmp': <built-in function ne>, 'cmpl': <built-in function lt>, 'div': <built-in function truediv>, 'mul': <built-in function mul>, 'or': <built-in function or_>, 'rem': <built-in function imod>, 'shl': <built-in function lshift>, 'shr': <built-in function rshift>, 'sub': <built-in function sub>, 'xor': <built-in function xor>}**

### angr.engines.soot.expressions.cast

class angr.engines.soot.expressions.cast.**SimSootExpr_Cast**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.condition

class angr.engines.soot.expressions.condition.**SimSootExpr_Condition**

Bases: *SimSootExpr*

**condition_str_to_function = {'eq': <built-in function eq>, 'ge': <built-in function SGE>, 'gt': <built-in function SGT>, 'le': <built-in function SLE>, 'lt': <built-in function SLT>, 'ne': <built-in function ne>}**

### angr.engines.soot.expressions.constants

class angr.engines.soot.expressions.constants.**SimSootExpr_IntConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.constants.**SimSootExpr_LongConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.constants.**SimSootExpr_FloatConstant**

Bases: *SimSootExpr*

class angr.engines.soot.expressions.constants.**SimSootExpr_DoubleConstant**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.constants.**SimSootExpr_StringConstant**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.constants.**SimSootExpr_ClassConstant**

Bases: *SimSootExpr*

**class** angr.engines.soot.expressions.constants.**SimSootExpr_NullConstant**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.instanceOf

**class** angr.engines.soot.expressions.instanceOf.**SimSootExpr_InstanceOf**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.instancefieldref

**class** angr.engines.soot.expressions.instancefieldref.**SimSootExpr_InstanceFieldRef**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.invoke

**class** angr.engines.soot.expressions.invoke.**InvokeBase**

Bases: *SimSootExpr*

**__init__**(*expr*, *state*)

**class** angr.engines.soot.expressions.invoke.**SimSootExpr_VirtualInvoke**

Bases: *InvokeBase*

Instance methods are dynamically resolved by the actual type of the base object (i.e. the object on which the method is invoked) and *not* by the type of the variable storing the reference.

**class** angr.engines.soot.expressions.invoke.**SimSootExpr_SpecialInvoke**

Bases: *InvokeBase*

Special invocations are used either for invoking instance methods of a superclass or a constructor. Compared to virtual invokes, the class containing the method is passed explicitly in the invoke expression (@expr.class_name) rather than determined dynamically by the type of the base objects.

**class** angr.engines.soot.expressions.invoke.**SimSootExpr_StaticInvoke**

Bases: *InvokeBase*

**class** angr.engines.soot.expressions.invoke.**SimSootExpr_InterfaceInvoke**

Bases: *SimSootExpr_VirtualInvoke*

### angr.engines.soot.expressions.length

**class** angr.engines.soot.expressions.length.**SimSootExpr_Length**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.local

**class** angr.engines.soot.expressions.local.**SimSootExpr_Local**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.new

class angr.engines.soot.expressions.new.**SimSootExpr_New**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.newArray

class angr.engines.soot.expressions.newArray.**SimSootExpr_NewArray**

Bases: *SimSootExpr*

**static new_array**(*state*, *element_type*, *size*)

Allocates a new array in memory and returns the reference to the base.

### angr.engines.soot.expressions.newMultiArray

class angr.engines.soot.expressions.newMultiArray.**SimSootExpr_NewMultiArray**

Bases: *SimSootExpr*

**static new_array**(*state*, *element_type*, *size*, *default_value_generator=None*)

Allocates a new multi array in memory and returns the reference to the base.

### angr.engines.soot.expressions.paramref

class angr.engines.soot.expressions.paramref.**SimSootExpr_ParamRef**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.phi

class angr.engines.soot.expressions.phi.**SimSootExpr_Phi**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.staticfieldref

class angr.engines.soot.expressions.staticfieldref.**SimSootExpr_StaticFieldRef**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.thisref

class angr.engines.soot.expressions.thisref.**SimSootExpr_ThisRef**

Bases: *SimSootExpr*

### angr.engines.soot.expressions.unsupported

class angr.engines.soot.expressions.unsupported.**SimSootExpr_Unsupported**

Bases: *SimSootExpr*

### angr.engines.soot.field_dispatcher

angr.engines.soot.field_dispatcher.**resolve_field**(*state*, *field_class*, *field_name*, *field_type*, *raise_exception_if_not_found=False*)

### angr.engines.soot.method_dispatcher

angr.engines.soot.method_dispatcher.**resolve_method**(*state*, *method_name*, *class_name*, *params=()*, *ret_type=None*, *include_superclasses=True*, *init_class=True*, *raise_exception_if_not_found=False*)

Resolves the method based on the given characteristics (name, class and params) The method may be defined in one of the superclasses of the given class (TODO: support interfaces).

**Return type**
archinfo.arch_soot.SootMethodDescriptor

### angr.engines.soot.statements

**class** angr.engines.soot.statements.**SimSootStmt_Assign**

Bases: *SimSootStmt*

**class** angr.engines.soot.statements.**SimSootStmt_Goto**

Bases: *SimSootStmt*

**class** angr.engines.soot.statements.**SimSootStmt_Identity**

Bases: *SimSootStmt*

**class** angr.engines.soot.statements.**SimSootStmt_If**

Bases: *SimSootStmt*

**class** angr.engines.soot.statements.**SimSootStmt_Invoke**

Bases: *SimSootStmt*

**class** angr.engines.soot.statements.**SimSootStmt_LookupSwitch**

Bases: *SwitchBase*

**class** angr.engines.soot.statements.**SimSootStmt_Return**

Bases: *SimSootStmt*

**__init__**(*stmt*, *state*)

**class** angr.engines.soot.statements.**SimSootStmt_ReturnVoid**

Bases: *SimSootStmt*

**__init__**(*stmt*, *state*)

**class** angr.engines.soot.statements.**SimSootStmt_TableSwitch**

Bases: *SwitchBase*

**class** angr.engines.soot.statements.**SimSootStmt_Throw**

Bases: *SimSootStmt*

angr.engines.soot.statements.**translate_stmt**(*stmt*, *state*)

### Submodules

*assign*
*base*
*goto*
*identity*

continues on next page

Table 54 – continued from previous page

| |
|---|
| *if_* |
| *invoke* |
| *return_* |
| *switch* |
| *throw* |

### angr.engines.soot.statements.assign

**class** angr.engines.soot.statements.assign.**SimSootStmt_Assign**

Bases: *SimSootStmt*

### angr.engines.soot.statements.base

**class** angr.engines.soot.statements.base.**SimSootStmt**

Bases: object

The base class of all Soot statements.

**__init__**(*stmt*, *state*)

**process**()

Process the statement and apply all effects on the state. :return: None

**property has_jump_targets**

**property has_invoke_target**

### angr.engines.soot.statements.goto

**class** angr.engines.soot.statements.goto.**SimSootStmt_Goto**

Bases: *SimSootStmt*

### angr.engines.soot.statements.identity

**class** angr.engines.soot.statements.identity.**SimSootStmt_Identity**

Bases: *SimSootStmt*

### angr.engines.soot.statements.if_

**class** angr.engines.soot.statements.if_.**SimSootStmt_If**

Bases: *SimSootStmt*

### angr.engines.soot.statements.invoke

**class** angr.engines.soot.statements.invoke.**SimSootStmt_Invoke**

Bases: *SimSootStmt*

### angr.engines.soot.statements.return_

**class** angr.engines.soot.statements.return_.**SimSootStmt_Return**

Bases: *SimSootStmt*

__init__(*stmt*, *state*)

class angr.engines.soot.statements.return_.SimSootStmt_ReturnVoid

Bases: *SimSootStmt*

__init__(*stmt*, *state*)

### angr.engines.soot.statements.switch

class angr.engines.soot.statements.switch.SwitchBase

Bases: *SimSootStmt*

class angr.engines.soot.statements.switch.SimSootStmt_TableSwitch

Bases: *SwitchBase*

class angr.engines.soot.statements.switch.SimSootStmt_LookupSwitch

Bases: *SwitchBase*

### angr.engines.soot.statements.throw

class angr.engines.soot.statements.throw.SimSootStmt_Throw

Bases: *SimSootStmt*

### angr.engines.soot.values

class angr.engines.soot.values.SimSootValue_ArrayBaseRef

Bases: *SimSootValue*

__init__(*heap_alloc_id*, *element_type*, *size*, *default_value_generator=None*)

id

element_type

size

type

get_default_value(*state*)

**Returns**
Default value for array elements.

add_default_value_generator(*generator*)

Add a generator for overwriting the default value for array elements.

**Parameters**
**generator** (function) – Function that given the state, returns a default value for array elements, e.g. *generator = lambda state: claripy.BVV(0, 32)*

classmethod from_sootvalue(*soot_value*, *state*)

class angr.engines.soot.values.SimSootValue_ArrayRef

Bases: *SimSootValue*

__init__(*base*, *index*)

id

**base**

**index**

**classmethod from_sootvalue**(*soot_value*, *state*)

**static translate_array_index**(*idx*, *state*)

**static check_array_bounds**(*idx*, *array*, *state*)

**class** angr.engines.soot.values.**SimSootValue_InstanceFieldRef**

Bases: *SimSootValue*

**__init__**(*heap_alloc_id*, *class_name*, *field_name*, *type_*)

**id**

**class_name**

**field_name**

**type**

**classmethod from_field_id**(*heap_alloc_id*, *field_id*)

**classmethod from_sootvalue**(*soot_value*, *state*)

**classmethod get_ref**(*state*, *obj_alloc_id*, *field_class_name*, *field_name*, *field_type*)

Resolve the field within the given state.

**class** angr.engines.soot.values.**SimSootValue_IntConstant**

Bases: *SimSootValue*

**__init__**(*value*, *type_*)

**value**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

**class** angr.engines.soot.values.**SimSootValue_Local**

Bases: *SimSootValue*

**__init__**(*name*, *type_*)

**id**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

**class** angr.engines.soot.values.**SimSootValue_ParamRef**

Bases: *SimSootValue*

**__init__**(*index*, *type_*)

**id**

**index**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

**class** angr.engines.soot.values.**SimSootValue_StaticFieldRef**

Bases: *SimSootValue*

**__init__**(*class_name*, *field_name*, *type_*)

**id**

**class_name**

**field_name**

**type**

**classmethod from_field_id**(*field_id*)

**classmethod from_sootvalue**(*soot_value*, *state*)

**classmethod get_ref**(*state*, *field_class_name*, *field_name*, *field_type*)

Resolve the field within the given state.

**class** angr.engines.soot.values.**SimSootValue_StringRef**

Bases: *SimSootValue*

**__init__**(*heap_alloc_id*)

**id**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

**static new_string**(*state*, *value*)

Allocate and initialize a new string in the context of the state passed.

The method returns the reference to the newly allocated string

**Parameters**

- **state** (SimState) – angr state where we want to allocate the string
- **value** (*claripy.String*) – value of the string to initialize

**Returns**

SimSootValue_StringRef

**class** angr.engines.soot.values.**SimSootValue_ThisRef**

Bases: *SimSootValue*

**__init__**(*heap_alloc_id*, *type_*, *symbolic=False*)

**property id**

**set_field**(*state*, *field_name*, *field_type*, *field_value*)

Sets an instance field.

**Parameters**

- **state** (SimState) – angr state where we want to allocate the object attribute

- **field_name** (*str*) – name of the attribute
- **field_type** (*str*) – type of the attribute
- **field_value** (SimSootValue) – attribute's value

**get_field**(*state*, *field_name*, *field_type*)

Gets the value of an instance field.

**Parameters**

- **state** (SimState) – angr state where we want to allocate the object attribute
- **field_name** (*str*) – name of the attribute
- **field_type** (*str*) – type of the attribute

**store_field**(*state*, *field_name*, *field_type*, *value*)

Store a field of a given object, WITHOUT RESOLVING HIERARCHY

**Parameters**

- **state** (SimState) – angr state where we want to allocate the object attribute
- **field_name** (*str*) – name of the attribute
- **field_value** (SimSootValue) – attribute's value

**load_field**(*state*, *field_name*, *field_type*)

Load a field of a given object, without resolving HIERARCHY

**Parameters**

- **tastate** (SimState) – angr state where we want to load the object attribute
- **field_name** (*str*) – name of the attribute
- **field_type** (*str*) – type of the attribute

**classmethod from_sootvalue**(*soot_value*, *state*)

**classmethod new_object**(*state*, *type_*, *symbolic=False*, *init_object=False*, *init_class=False*)

Creates a new object reference.

**Parameters**

- **state** – State associated to the object.
- **type_** – Class of the object.
- **init_object** – Whether the objects initializer method should be run.
- **init_class** – Whether the class initializer method should be run.

**Returns**

Reference to the new object.

angr.engines.soot.values.**translate_value**(*value*, *state*)

### Submodules

| |
|---|
| *arrayref* |
| *base* |
| *constants* |

continues on next page

Table 55 – continued from previous page

| |
|---|
| *instancefieldref* |
| *local* |
| *paramref* |
| *staticfieldref* |
| *strref* |
| *thisref* |

### angr.engines.soot.values.arrayref

**class** angr.engines.soot.values.arrayref.**SimSootValue_ArrayBaseRef**

Bases: *SimSootValue*

**__init__**(*heap_alloc_id*, *element_type*, *size*, *default_value_generator=None*)

**id**

**element_type**

**size**

**type**

**get_default_value**(*state*)

**Returns**
Default value for array elements.

**add_default_value_generator**(*generator*)

Add a generator for overwriting the default value for array elements.

**Parameters**
**generator** (function) – Function that given the state, returns a default value for array elements, e.g. *generator = lambda state: claripy.BVV(0, 32)*

**classmethod from_sootvalue**(*soot_value*, *state*)

**class** angr.engines.soot.values.arrayref.**SimSootValue_ArrayRef**

Bases: *SimSootValue*

**__init__**(*base*, *index*)

**id**

**base**

**index**

**classmethod from_sootvalue**(*soot_value*, *state*)

**static translate_array_index**(*idx*, *state*)

**static check_array_bounds**(*idx*, *array*, *state*)

### angr.engines.soot.values.base

class angr.engines.soot.values.base.**SimSootValue**

Bases: object

**classmethod from_sootvalue**(*soot_value*, *state*)

### angr.engines.soot.values.constants

class angr.engines.soot.values.constants.**SimSootValue_IntConstant**

Bases: *SimSootValue*

**__init__**(*value*, *type_*)

**value**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

### angr.engines.soot.values.instancefieldref

class angr.engines.soot.values.instancefieldref.**SimSootValue_InstanceFieldRef**

Bases: *SimSootValue*

**__init__**(*heap_alloc_id*, *class_name*, *field_name*, *type_*)

**id**

**class_name**

**field_name**

**type**

**classmethod from_field_id**(*heap_alloc_id*, *field_id*)

**classmethod from_sootvalue**(*soot_value*, *state*)

**classmethod get_ref**(*state*, *obj_alloc_id*, *field_class_name*, *field_name*, *field_type*)

Resolve the field within the given state.

### angr.engines.soot.values.local

class angr.engines.soot.values.local.**SimSootValue_Local**

Bases: *SimSootValue*

**__init__**(*name*, *type_*)

**id**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

### angr.engines.soot.values.paramref

class angr.engines.soot.values.paramref.**SimSootValue_ParamRef**

Bases: *SimSootValue*

**__init__**(*index*, *type_*)

**id**

**index**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

### angr.engines.soot.values.staticfieldref

class angr.engines.soot.values.staticfieldref.**SimSootValue_StaticFieldRef**

Bases: *SimSootValue*

**__init__**(*class_name*, *field_name*, *type_*)

**id**

**class_name**

**field_name**

**type**

**classmethod from_field_id**(*field_id*)

**classmethod from_sootvalue**(*soot_value*, *state*)

**classmethod get_ref**(*state*, *field_class_name*, *field_name*, *field_type*)

Resolve the field within the given state.

### angr.engines.soot.values.strref

class angr.engines.soot.values.strref.**SimSootValue_StringRef**

Bases: *SimSootValue*

**__init__**(*heap_alloc_id*)

**id**

**type**

**classmethod from_sootvalue**(*soot_value*, *state*)

**static new_string**(*state*, *value*)

Allocate and initialize a new string in the context of the state passed.

The method returns the reference to the newly allocated string

**Parameters**

- **state** (SimState) – angr state where we want to allocate the string
- **value** (*claripy.String*) – value of the string to initialize

**Returns**
SimSootValue_StringRef

### angr.engines.soot.values.thisref

**class** angr.engines.soot.values.thisref.**SimSootValue_ThisRef**

Bases: *SimSootValue*

**__init__**(*heap_alloc_id*, *type_*, *symbolic=False*)

**property id**

**set_field**(*state*, *field_name*, *field_type*, *field_value*)
Sets an instance field.

**Parameters**

- **state** (SimState) – angr state where we want to allocate the object attribute
- **field_name** (*str*) – name of the attribute
- **field_type** (*str*) – type of the attribute
- **field_value** (SimSootValue) – attribute's value

**get_field**(*state*, *field_name*, *field_type*)
Gets the value of an instance field.

**Parameters**

- **state** (SimState) – angr state where we want to allocate the object attribute
- **field_name** (*str*) – name of the attribute
- **field_type** (*str*) – type of the attribute

**store_field**(*state*, *field_name*, *field_type*, *value*)
Store a field of a given object, WITHOUT RESOLVING HIERARCHY

**Parameters**

- **state** (SimState) – angr state where we want to allocate the object attribute
- **field_name** (*str*) – name of the attribute
- **field_value** (SimSootValue) – attribute's value

**load_field**(*state*, *field_name*, *field_type*)
Load a field of a given object, without resolving HIERARCHY

**Parameters**

- **tastate** (SimState) – angr state where we want to load the object attribute
- **field_name** (*str*) – name of the attribute
- **field_type** (*str*) – type of the attribute

**classmethod from_sootvalue**(*soot_value*, *state*)

**classmethod new_object**(*state*, *type_*, *symbolic=False*, *init_object=False*, *init_class=False*)
Creates a new object reference.

**Parameters**

- **state** – State associated to the object.
- **type_** – Class of the object.
- **init_object** – Whether the objects initializer method should be run.
- **init_class** – Whether the class initializer method should be run.

**Returns**
Reference to the new object.

### 10.15.10 angr.engines.successors

**class** angr.engines.successors.**SimSuccessors**

Bases: object

This class serves as a categorization of all the kinds of result states that can come from a SimEngine run.

**Variables**

- **addr** (*int*) – The address at which execution is taking place, as a python int
- **initial_state** – The initial state for which execution produced these successors
- **engine** – The engine that produced these successors
- **sort** (str | None) – A string identifying the type of engine that produced these successors
- **processed** (*bool*) – Whether or not the processing succeeded
- **description** (*str*) – A textual description of the execution step

The successor states produced by this run are categorized into several lists:

**Variables**

- **artifacts** (*dict*) – Any analysis byproducts (for example, an IRSB) that were produced during execution
- **successors** (list[*HeavyState*]) – The "normal" successors. IP may be symbolic, but must have reasonable number of solutions
- **unsat_successors** (list[*HeavyState*]) – Any successor which is unsatisfiable after its guard condition is added.
- **all_successors** (list[*HeavyState*]) – successors + unsat_successors
- **flat_successors** (list[*HeavyState*]) – The normal successors, but any symbolic IPs have been concretized. There is one state in this list for each possible value an IP may be concretized to for each successor state.
- **unconstrained_successors** (list[*HeavyState*]) – Any state for which during the flattening process we find too many solutions.

A more detailed description of the successor lists may be found here: https://docs.angr.io/core-concepts/simulation#simsuccessors

**__init__**(*addr*, *initial_state*)

**Parameters**

- **addr** (*int* | *SootAddressDescriptor* | *None*)
- **initial_state** (HeavyState | *None*)

property is_empty

add_successor(*state*, *target*, *guard*, *jumpkind*, *add_guard=True*, *exit_stmt_idx=None*, *exit_ins_addr=None*, *source=None*)

Add a successor state of the SimRun. This procedure stores method parameters into state.scratch, does some housekeeping, and calls out to helper functions to prepare the state and categorize it into the appropriate successor lists.

**Parameters**

- **state** (SimState) – The successor state.
- **target** – The target (of the jump/call/ret).
- **guard** – The guard expression.
- **jumpkind** (*str*) – The jumpkind (call, ret, jump, or whatnot).
- **add_guard** (*bool*) – Whether to add the guard constraint (default: True).
- **exit_stmt_idx** (*int*) – The ID of the exit statement, an integer by default. 'default' stands for the default exit, and None means it's not from a statement (for example, from a SimProcedure).
- **exit_ins_addr** (*int*) – The instruction pointer of this exit, which is an integer by default.
- **source** (*int*) – The source of the jump (i.e., the address of the basic block).

type angr.engines.successors.HeavyState = *SimState*[int | SootAddressDescriptor, claripy.ast.BV | SootAddressDescriptor]

class angr.engines.successors.SuccessorsEngine

Bases: *SimEngine*[*HeavyState*, *SimSuccessors*]

A mixin for SimEngine which implements process to perform common operations related to symbolic execution and dispatches to a process_successors method to fill a SimSuccessors object with the results.

__init__(*project*)

**Parameters**
**project** (Project)

process(*state*, ***kwargs*)

Perform execution with a state.

You should only override this method in a subclass in order to provide the correct method signature and docstring. You should override the _process method to do your actual execution.

**Parameters**

- **state** (*HeavyState*) – The state with which to execute. This state will be copied before modification.
- **inline** – This is an inline execution. Do not bother copying the state.
- **force_addr** – Force execution to pretend that we're working at this concrete address

**Return type**
*SimSuccessors*

**Returns**
A SimSuccessors object categorizing the execution's successor states

**process_successors**(*successors*, ***kwargs*)

Implement this function to fill out the SimSuccessors object with the results of stepping state.

In order to implement a model where multiple mixins can potentially handle a request, a mixin may implement this method and then perform a super() call if it wants to pass on handling to the next mixin.

Keep in mind python's method resolution order when composing multiple classes implementing this method. In short: left-to-right, depth-first, but deferring any base classes which are shared by multiple subclasses (the merge point of a diamond pattern in the inheritance graph) until the last point where they would be encountered in this depth-first search. For example, if you have classes A, B(A), C(B), D(A), E(C, D), then the method resolution order will be E, C, B, D, A.

**Parameters**

- **state** – The state to manipulate
- **successors** (*SimSuccessors*) – The successors object to fill out
- **kwargs** – Any extra arguments. Do not fail if you are passed unexpected arguments.

### 10.15.11 angr.engines.syscall

**class** angr.engines.syscall.**SimEngineSyscall**

Bases: *SuccessorsEngine*, *ProcedureMixin*

A SimEngine mixin which adds a successors handling step that checks if a syscall was just requested and if so handles it as a step.

### 10.15.12 angr.engines.unicorn

**class** angr.engines.unicorn.**SimEngineUnicorn**

Bases: *SuccessorsEngine*

Concrete execution in the Unicorn Engine, a fork of qemu.

Responds to the following parameters in the step stack:

- step: How many basic blocks we want to execute
- extra_stop_points: A collection of addresses at which execution should halt

**__init__**(*project*)

**Parameters**
**project** (Project)

### 10.15.13 angr.engines.vex

**class** angr.engines.vex.**ClaripyDataMixin**

Bases: *VEXMixin*

This mixin provides methods that makes the vex engine process guest code using claripy ASTs as the data domain.

**class** angr.engines.vex.**HeavyResilienceMixin**

Bases: *VEXResilienceMixin*, *ClaripyDataMixin*

**class** angr.engines.vex.**HeavyVEXMixin**

Bases: *SuccessorsEngine*, *ClaripyDataMixin*, *SimStateStorageMixin*, *VEXMixin*, *VEXLifter*

Execution engine based on VEX, Valgrind's IR.

Responds to the following parameters to the step stack:

- irsb: The PyVEX IRSB object to use for execution. If not provided one will be lifted.
- skip_stmts: The number of statements to skip in processing
- last_stmt: Do not execute any statements after this statement
- whitelist: Only execute statements in this set
- thumb: Whether the block should be force to be lifted in ARM's THUMB mode.
- **extra_stop_points:**
  An extra set of points at which to break basic blocks
- opt_level: The VEX optimization level to use.
- insn_bytes: A string of bytes to use for the block instead of the project.
- size: The maximum size of the block, in bytes.
- num_inst: The maximum number of instructions.
- traceflags: traceflags to be passed to VEX. (default: 0)

**class** angr.engines.vex.**SimInspectMixin**

Bases: *VEXMixin*

**handle_vex_block**(*irsb*)

**class** angr.engines.vex.**SuperFastpathMixin**

Bases: *VEXSlicingMixin*

This mixin implements the superfastpath execution mode, which skips all but the last four instructions.

**handle_vex_block**(*irsb*)

**class** angr.engines.vex.**TrackActionsMixin**

Bases: *HeavyVEXMixin*

**__init__**(**args*, ***kwargs*)

**handle_vex_block**(*irsb*)

**class** angr.engines.vex.**VEXLifter**

Bases: *SimEngine*

Implements the VEX lifter engine mixin.

**__init__**(*project*, *use_cache=None*, *cache_size=5000*, *default_opt_level=1*, *selfmodifying_code=None*, *single_step=False*, *default_strict_block_end=False*, ***kwargs*)

**clear_cache**()

**lift_vex**(*addr=None*, *state=None*, *clemory=None*, *insn_bytes=None*, *offset=None*, *arch=None*, *size=None*, *num_inst=None*, *traceflags=0*, *thumb=False*, *extra_stop_points=None*, *opt_level=None*, *strict_block_end=None*, *skip_stmts=False*, *collect_data_refs=False*, *cross_insn_opt=None*, *load_from_ro_regions=False*, *const_prop=False*)

Lift an IRSB.

There are many possible valid sets of parameters. You at the very least must pass some source of data, some source of an architecture, and some source of an address.

Sources of data in order of priority: insn_bytes, clemory, state

Sources of an address, in order of priority: addr, state

Sources of an architecture, in order of priority: arch, clemory, state

**Parameters**

- **state** – A state to use as a data source.
- **clemory** (Clemory | ClemoryReadOnlyView | None) – A cle.memory.Clemory object to use as a data source.
- **addr** – The address at which to start the block.
- **thumb** – Whether the block should be lifted in ARM's THUMB mode.
- **opt_level** – The VEX optimization level to use. The final IR optimization level is determined by (ordered by priority): - Argument opt_level - opt_level is set to 1 if OPTIMIZE_IR exists in state options - self._default_opt_level
- **insn_bytes** (bytes | None) – A string of bytes to use as a data source.
- **offset** – If using insn_bytes, the number of bytes in it to skip over.
- **size** – The maximum size of the block, in bytes.
- **num_inst** – The maximum number of instructions.
- **traceflags** – traceflags to be passed to VEX. (default: 0)
- **strict_block_end** – Whether to force blocks to end at all conditional branches (default: false)

**Return type**
IRSB

**class** angr.engines.vex.**VEXMixin**

Bases: *SimEngine*

**__init__**(*project*, ***kwargs*)

**handle_vex_block**(*irsb*)

**Parameters**
**irsb** (*IRSB*)

**class** angr.engines.vex.**VEXResilienceMixin**

Bases: *VEXMixin*

**class** angr.engines.vex.**VEXSlicingMixin**

Bases: *VEXMixin*

**__init__**(**args*, ***kwargs*)

**process**(*state*, *block=None*, *skip_stmts=0*, *last_stmt=None*, *whitelist=None*, ***kwargs*)

**handle_vex_block**(*irsb*)

## Submodules

*claripy*
*heavy*
*lifter*
*light*

### angr.engines.vex.claripy

**class** angr.engines.vex.claripy.**ClaripyDataMixin**

Bases: *VEXMixin*

This mixin provides methods that makes the vex engine process guest code using claripy ASTs as the data domain.

### Submodules

| | |
|---|---|
| *ccall* | |
| *datalayer* | |
| *irop* | This module contains symbolic implementations of VEX operations. |

### angr.engines.vex.claripy.ccall

angr.engines.vex.claripy.ccall.**calc_paritybit**(*p*, *msb=7*, *lsb=0*)

angr.engines.vex.claripy.ccall.**calc_zerobit**(*p*)

angr.engines.vex.claripy.ccall.**boolean_extend**(*O*, *a*, *b*, *size*)

angr.engines.vex.claripy.ccall.**op_concretize**(*op*)

angr.engines.vex.claripy.ccall.**strip_simaction**(*val*)

**exception** angr.engines.vex.claripy.ccall.**CCallMultivaluedException**

Bases: Exception

**type** angr.engines.vex.claripy.ccall.**Platform** = Literal['AMD64', 'X86']

**class** angr.engines.vex.claripy.ccall.**CCallTables**

Bases: TypedDict

The condition-code lookup tables VEX's x86/amd64 ccall helpers are defined against.

**size:** int

**CondTypes:** dict[str, int]

**CondBitOffsets:** dict[str, int]

**CondBitMasks:** dict[str, int]

**OpTypes:** dict[str, int | None]

angr.engines.vex.claripy.ccall.**pc_preamble**(*nbits*)

angr.engines.vex.claripy.ccall.**pc_make_rdata**(*nbits*, *cf*, *pf*, *af*, *zf*, *sf*, *of*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_make_rdata_if_necessary**(*nbits*, *cf*, *pf*, *af*, *zf*, *sf*, *of*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_ADD**(*state*, *nbits*, *arg_l*, *arg_r*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_SUB**(*state*, *nbits*, *arg_l*, *arg_r*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_LOGIC**(*state*, *nbits*, *arg_l*, *arg_r*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_DEC**(*state*, *nbits*, *res*, *_*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_ADC**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_ADCX**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *is_adc*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_ANDN**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_BLSI**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_BLSMSK**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_BLSR**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_SBB**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_INC**(*state*, *nbits*, *res*, *_*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_SHL**(*state*, *nbits*, *remaining*, *shifted*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_SHR**(*state*, *nbits*, *remaining*, *shifted*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_ROL**(*state*, *nbits*, *res*, *_*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_ROR**(*state*, *nbits*, *res*, *_*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_UMUL**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_UMULQ**(**args*, ***kwargs*)

angr.engines.vex.claripy.ccall.**pc_actions_SMUL**(*state*, *nbits*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_SMULQ**(**args*, ***kwargs*)

angr.engines.vex.claripy.ccall.**pc_calculate_rdata_all_WRK**(*state*, *cc_op*, *cc_dep1_formal*, *cc_dep2_formal*, *cc_ndep_formal*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_calculate_rdata_all**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_calculate_condition**(*state*, *cond*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_actions_op_SUB**(*arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_op_DEC**(*arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_op_INC**(*arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_op_SHR**(*arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_op_SHL**(*arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_op_ADD**(*arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_op_LOGIC**(*arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondZ**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondNZ**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondS**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondB**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondBE**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondNBE**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondL**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondLE**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_cond_CondNLE**(*state*, *cc_expr*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondZ**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondNZ**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondB**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondBE**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondNBE**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondL**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondLE**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_actions_SUB_CondNLE**(*state*, *arg_l*, *arg_r*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**pc_calculate_condition_simple**(*state*, *cond*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

A simplified version of pc_calculate_condition(). Please refer to the documentation of Simplified CCalls above.

Limitation: symbolic flags are not supported for now.

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**pc_calculate_rdata_c**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*, *platform*)

**Parameters**
**platform** (Platform)

angr.engines.vex.claripy.ccall.**generic_rotate_with_carry**(*state*, *left*, *arg*, *rot_amt*, *carry_bit_in*, *sz*)

angr.engines.vex.claripy.ccall.**amd64g_check_ldmxcsr**(*state*, *mxcsr*)

angr.engines.vex.claripy.ccall.**amd64g_create_mxcsr**(*state*, *sseround*)

angr.engines.vex.claripy.ccall.**amd64g_check_fldcw**(*state*, *fpucw*)

angr.engines.vex.claripy.ccall.**amd64g_create_fpucw**(*state*, *fpround*)

angr.engines.vex.claripy.ccall.**amd64g_calculate_RCL**(*state*, *arg*, *rot_amt*, *eflags_in*, *sz*)

angr.engines.vex.claripy.ccall.**amd64g_calculate_RCR**(*state*, *arg*, *rot_amt*, *eflags_in*, *sz*)

angr.engines.vex.claripy.ccall.**amd64g_calculate_mmx_pmaddwd**(*_state*, *xx*, *yy*)

angr.engines.vex.claripy.ccall.**amd64g_calculate_condition**(*state*, *cond*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**amd64g_calculate_rflags_all**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**amd64g_calculate_rflags_c**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**x86g_calculate_RCL**(*state*, *arg*, *rot_amt*, *eflags_in*, *sz*)

angr.engines.vex.claripy.ccall.**x86g_calculate_RCR**(*state*, *arg*, *rot_amt*, *eflags_in*, *sz*)

angr.engines.vex.claripy.ccall.**x86g_calculate_condition**(*state*, *cond*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**x86g_calculate_eflags_all**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**x86g_calculate_eflags_c**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_ndep*)

angr.engines.vex.claripy.ccall.**x86g_check_fldcw**(*state*, *fpucw*)

angr.engines.vex.claripy.ccall.**x86g_create_fpucw**(*state*, *fpround*)

angr.engines.vex.claripy.ccall.**x86g_calculate_daa_das_aaa_aas**(*state*, *flags_and_AX*, *opcode*)

angr.engines.vex.claripy.ccall.**x86g_calculate_aad_aam**(*state*, *flags_and_AX*, *opcode*)

angr.engines.vex.claripy.ccall.**get_segdescr_base**(*state*, *descriptor*)

angr.engines.vex.claripy.ccall.**get_segdescr_limit**(*state*, *descriptor*)

angr.engines.vex.claripy.ccall.**x86g_use_seg_selector**(*state*, *ldt*, *gdt*, *seg_selector*, *virtual_addr*)

angr.engines.vex.claripy.ccall.**armg_calculate_flag_n**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm_zerobit**(*state*, *x*)

angr.engines.vex.claripy.ccall.**armg_calculate_flag_z**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**armg_calculate_flag_c**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**armg_calculate_flag_v**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**armg_calculate_flags_nzcv**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**armg_calculate_condition**(*state*, *cond_n_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm64g_calculate_flag_n**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm64_zerobit**(*state*, *x*)

angr.engines.vex.claripy.ccall.**u64_to_u32**(*n*)

angr.engines.vex.claripy.ccall.**arm64g_32bit_truncate_operands**(*cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm64g_calculate_flag_z**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm64g_calculate_flag_c**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm64g_calculate_flag_v**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm64g_calculate_flags_nzcv**(*state*, *cc_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**arm64g_calculate_condition**(*state*, *cond_n_op*, *cc_dep1*, *cc_dep2*, *cc_dep3*)

angr.engines.vex.claripy.ccall.**riscv64g_calculate_fflags_feq_s**(*state*, *a1*, *a2*)

angr.engines.vex.claripy.ccall.**riscv64g_calculate_fflags_flt_s**(*state*, *a1*, *a2*)

angr.engines.vex.claripy.ccall.**riscv64g_calculate_fflags_fle_s**(*state*, *a1*, *a2*)

angr.engines.vex.claripy.ccall.**riscv64g_calculate_fflags_feq_d**(*state*, *a1*, *a2*)

angr.engines.vex.claripy.ccall.**riscv64g_calculate_fflags_flt_d**(*state*, *a1*, *a2*)

angr.engines.vex.claripy.ccall.**riscv64g_calculate_fflags_fle_d**(*state*, *a1*, *a2*)

### angr.engines.vex.claripy.datalayer

angr.engines.vex.claripy.datalayer.**value**(*ty*, *val*, *size=None*)

**Parameters**

- **ty** (*str*)
- **val** (*float*)
- **size** (*int | None*)

angr.engines.vex.claripy.datalayer.**symbol**(*ty*, *name*)

**class** angr.engines.vex.claripy.datalayer.**ClaripyDataMixin**

Bases: *VEXMixin*

This mixin provides methods that makes the vex engine process guest code using claripy ASTs as the data domain.

### angr.engines.vex.claripy.irop

This module contains symbolic implementations of VEX operations.

angr.engines.vex.claripy.irop.**op_attrs**(*p*)

angr.engines.vex.claripy.irop.**make_operations**()

angr.engines.vex.claripy.irop.**supports_vector**(*f*)

class angr.engines.vex.claripy.irop.**SimIROp**

Bases: object

A symbolic version of a Vex IR operation.

**__init__**(*name*, ***attrs*)

**name**

**op_attrs**

**calculate**(**args*)

**extend_size**(*o*)

**property is_signed**

**NO_RM = {'Abs', 'Neg'}**

**vector_args**(*args*)

Yields each of the individual lane pairs from the arguments, in order from most significan to least significant

**generic_minmax**(*args*, *cmp_op*)

**generic_compare**(*args*, *comparison*)

**generic_shift_thing**(*args*, *op*)

**static pow**(*rm*, *arg*, *n*)

angr.engines.vex.claripy.irop.**vexop_to_simop**(*op*, *extended=True*, *fp=True*)

### angr.engines.vex.heavy

class angr.engines.vex.heavy.**HeavyResilienceMixin**

Bases: *VEXResilienceMixin*, *ClaripyDataMixin*

class angr.engines.vex.heavy.**HeavyVEXMixin**

Bases: *SuccessorsEngine*, *ClaripyDataMixin*, *SimStateStorageMixin*, *VEXMixin*, *VEXLifter*

Execution engine based on VEX, Valgrind's IR.

Responds to the following parameters to the step stack:

- irsb: The PyVEX IRSB object to use for execution. If not provided one will be lifted.
- skip_stmts: The number of statements to skip in processing
- last_stmt: Do not execute any statements after this statement
- whitelist: Only execute statements in this set
- thumb: Whether the block should be force to be lifted in ARM's THUMB mode.
- **extra_stop_points:**
  An extra set of points at which to break basic blocks
- opt_level: The VEX optimization level to use.
- insn_bytes: A string of bytes to use for the block instead of the project.
- size: The maximum size of the block, in bytes.
- num_inst: The maximum number of instructions.

- traceflags: traceflags to be passed to VEX. (default: 0)

class angr.engines.vex.heavy.**SimInspectMixin**

Bases: *VEXMixin*

**handle_vex_block**(*irsb*)

class angr.engines.vex.heavy.**SuperFastpathMixin**

Bases: *VEXSlicingMixin*

This mixin implements the superfastpath execution mode, which skips all but the last four instructions.

**handle_vex_block**(*irsb*)

class angr.engines.vex.heavy.**TrackActionsMixin**

Bases: *HeavyVEXMixin*

**__init__**(**args*, ***kwargs*)

**handle_vex_block**(*irsb*)

### Submodules

| |
|---|
| *actions* |
| *concretizers* |
| *dirty* |
| *heavy* |
| *inspect* |
| *resilience* |
| *super_fastpath* |

### angr.engines.vex.heavy.actions

class angr.engines.vex.heavy.actions.**TrackActionsMixin**

Bases: *HeavyVEXMixin*

**__init__**(**args*, ***kwargs*)

**handle_vex_block**(*irsb*)

### angr.engines.vex.heavy.concretizers

angr.engines.vex.heavy.concretizers.**translate_rm**(*rm_num*)

angr.engines.vex.heavy.concretizers.**concretize_2xm1**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_abs_float64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_add_float64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_add32f04**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_add64f02**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_cmpf64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_divf64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_div32f04**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_div64f02**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_float32_to_float64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_float64_to_float32**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_float64_to_int64s**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_int32s_to_float64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_int64s_to_float64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_mulf64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_mul32f04**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_mul64f02**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_fscale**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_fsqrt32**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_fsqrt**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_prem**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_prem_flags**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_reinterp_float64_as_int64**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_sub32f04**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_sub64f02**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_trig_cos**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_trig_sin**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_trig_tan**(*state*, *args*)

angr.engines.vex.heavy.concretizers.**concretize_yl2x**(*state*, *args*)

### angr.engines.vex.heavy.dirty

angr.engines.vex.heavy.dirty.**ppcg_dirtyhelper_MFTB**(*state*)

angr.engines.vex.heavy.dirty.**ppc32g_dirtyhelper_MFSPR_287**(*state*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_RDTSC**(*state*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_RDTSC**(*state*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_CPUID_baseline**(*state*, _)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_CPUID_avx_and_cx16**(*state*, *_*, *hasF16C=None*, *hasRDRAND=None*, *hasRDSEED=None*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_CPUID_avx2**(*state*, *_*, *hasF16C=None*, *hasRDRAND=None*, *hasRDSEED=None*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_CPUID_avx512_KNL**(*state*, *_*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_CPUID_avx512_SKX**(*state*, *_*)

angr.engines.vex.heavy.dirty.**amd64g_create_mxcsr**(*_*, *sseround*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_XSAVE_COMPONENT_1_EXCLUDING_XMMREGS**(*state*, *_*, *addr*)

angr.engines.vex.heavy.dirty.**amd64g_check_ldmxcsr**(*state*, *mxcsr*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_XRSTOR_COMPONENT_1_EXCLUDING_XMMREGS**(*state*, *_*, *addr*)

angr.engines.vex.heavy.dirty.**CORRECT_amd64g_dirtyhelper_CPUID_avx_and_cx16**(*state*, *_*, *hasF16C=None*, *hasR-DRAND=None*, *hasRD-SEED=None*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_IN**(*state*, *portno*, *sz*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_OUT**(*state*, *portno*, *data*, *sz*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_SxDT**(*state*, *addr*, *op*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_CPUID_sse0**(*state*, *_*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_CPUID_sse2**(*state*, *_*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_CPUID_sse3**(*state*, *_*)

angr.engines.vex.heavy.dirty.**CORRECT_x86g_dirtyhelper_CPUID_sse2**(*state*, *_*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_IN**(*state*, *portno*, *sz*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_OUT**(*state*, *portno*, *data*, *sz*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_SxDT**(*state*, *addr*, *op*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_LGDT_LIDT**(*state*, *addr*, *op*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_FINIT**(*state*, *gsptr*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_FINIT**(*state*, *gsptr*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_write_cr0**(*state*, *value*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_loadF80le**(*state*, *addr*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_storeF80le**(*state*, *addr*, *qword*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_RDMSR**(*state*, *msr*)

angr.engines.vex.heavy.dirty.**x86g_dirtyhelper_XGETBV**(*state*, *reg*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_RDMSR**(*state*, *msr*)

angr.engines.vex.heavy.dirty.**amd64g_dirtyhelper_XGETBV**(*state*, *reg*)

angr.engines.vex.heavy.dirty.**arm64g_dirtyhelper_MRS_DCZID_EL0**(*state*)

angr.engines.vex.heavy.dirty.**mips_dirtyhelper_rdhwr**(*state*, *rd*)

### angr.engines.vex.heavy.heavy

**exception** angr.engines.vex.heavy.heavy.**VEXEarlyExit**

Bases: Exception

**class** angr.engines.vex.heavy.heavy.**SimStateStorageMixin**

Bases: *VEXMixin*

**class** angr.engines.vex.heavy.heavy.**HeavyVEXMixin**

Bases: *SuccessorsEngine*, *ClaripyDataMixin*, *SimStateStorageMixin*, *VEXMixin*, *VEXLifter*

Execution engine based on VEX, Valgrind's IR.

Responds to the following parameters to the step stack:

- irsb: The PyVEX IRSB object to use for execution. If not provided one will be lifted.
- skip_stmts: The number of statements to skip in processing
- last_stmt: Do not execute any statements after this statement
- whitelist: Only execute statements in this set
- thumb: Whether the block should be force to be lifted in ARM's THUMB mode.
- **extra_stop_points:**
  An extra set of points at which to break basic blocks
- opt_level: The VEX optimization level to use.
- insn_bytes: A string of bytes to use for the block instead of the project.
- size: The maximum size of the block, in bytes.
- num_inst: The maximum number of instructions.
- traceflags: traceflags to be passed to VEX. (default: 0)

### angr.engines.vex.heavy.inspect

**class** angr.engines.vex.heavy.inspect.**SimInspectMixin**

Bases: *VEXMixin*

**handle_vex_block**(*irsb*)

### angr.engines.vex.heavy.resilience

**class** angr.engines.vex.heavy.resilience.**HeavyResilienceMixin**

Bases: *VEXResilienceMixin*, *ClaripyDataMixin*

### angr.engines.vex.heavy.super_fastpath

**class** angr.engines.vex.heavy.super_fastpath.**SuperFastpathMixin**

Bases: *VEXSlicingMixin*

This mixin implements the superfastpath execution mode, which skips all but the last four instructions.

**handle_vex_block**(*irsb*)

### angr.engines.vex.lifter

**class** angr.engines.vex.lifter.**VEXLifter**

Bases: *SimEngine*

Implements the VEX lifter engine mixin.

**__init__**(*project*, *use_cache=None*, *cache_size=5000*, *default_opt_level=1*, *selfmodifying_code=None*, *single_step=False*, *default_strict_block_end=False*, ***kwargs*)

**clear_cache**()

**lift_vex**(*addr=None*, *state=None*, *clemory=None*, *insn_bytes=None*, *offset=None*, *arch=None*, *size=None*, *num_inst=None*, *traceflags=0*, *thumb=False*, *extra_stop_points=None*, *opt_level=None*, *strict_block_end=None*, *skip_stmts=False*, *collect_data_refs=False*, *cross_insn_opt=None*, *load_from_ro_regions=False*, *const_prop=False*)

Lift an IRSB.

There are many possible valid sets of parameters. You at the very least must pass some source of data, some source of an architecture, and some source of an address.

Sources of data in order of priority: insn_bytes, clemory, state

Sources of an address, in order of priority: addr, state

Sources of an architecture, in order of priority: arch, clemory, state

**Parameters**

- **state** – A state to use as a data source.
- **clemory** (Clemory | ClemoryReadOnlyView | None) – A cle.memory.Clemory object to use as a data source.
- **addr** – The address at which to start the block.
- **thumb** – Whether the block should be lifted in ARM's THUMB mode.
- **opt_level** – The VEX optimization level to use. The final IR optimization level is determined by (ordered by priority): - Argument opt_level - opt_level is set to 1 if OPTIMIZE_IR exists in state options - self._default_opt_level
- **insn_bytes** (bytes | None) – A string of bytes to use as a data source.
- **offset** – If using insn_bytes, the number of bytes in it to skip over.
- **size** – The maximum size of the block, in bytes.
- **num_inst** – The maximum number of instructions.

- **traceflags** – traceflags to be passed to VEX. (default: 0)
- **strict_block_end** – Whether to force blocks to end at all conditional branches (default: false)

**Return type**
IRSB

### angr.engines.vex.light

**class** angr.engines.vex.light.**VEXMixin**

Bases: *SimEngine*

**__init__**(*project*, ***kwargs*)

**handle_vex_block**(*irsb*)

**Parameters**
**irsb** (*IRSB*)

**class** angr.engines.vex.light.**VEXResilienceMixin**

Bases: *VEXMixin*

**class** angr.engines.vex.light.**VEXSlicingMixin**

Bases: *VEXMixin*

**__init__**(**args*, ***kwargs*)

**process**(*state*, *block=None*, *skip_stmts=0*, *last_stmt=None*, *whitelist=None*, ***kwargs*)

**handle_vex_block**(*irsb*)

### Submodules

*light*
*resilience*
*slicing*

### angr.engines.vex.light.light

**class** angr.engines.vex.light.light.**VEXMixin**

Bases: *SimEngine*

**__init__**(*project*, ***kwargs*)

**handle_vex_block**(*irsb*)

**Parameters**
**irsb** (*IRSB*)

### angr.engines.vex.light.resilience

**class** angr.engines.vex.light.resilience.**VEXResilienceMixin**

Bases: *VEXMixin*

### angr.engines.vex.light.slicing

**class** angr.engines.vex.light.slicing.**VEXSlicingMixin**

Bases: *VEXMixin*

**__init__**(**args*, ***kwargs*)

**process**(*state*, *block=None*, *skip_stmts=0*, *last_stmt=None*, *whitelist=None*, ***kwargs*)

**handle_vex_block**(*irsb*)

## 10.16 angr.errors

**exception** angr.errors.**AngrError**

Bases: Exception

**exception** angr.errors.**AngrRuntimeError**

Bases: RuntimeError

**exception** angr.errors.**AngrValueError**

Bases: *AngrError*, ValueError

**exception** angr.errors.**AngrLifterError**

Bases: *AngrError*

**exception** angr.errors.**AngrExitError**

Bases: *AngrError*

**exception** angr.errors.**AngrPathError**

Bases: *AngrError*

**exception** angr.errors.**AngrVaultError**

Bases: *AngrError*

**exception** angr.errors.**PathUnreachableError**

Bases: *AngrPathError*

**exception** angr.errors.**SimulationManagerError**

Bases: *AngrError*

**exception** angr.errors.**AngrInvalidArgumentError**

Bases: *AngrError*

**exception** angr.errors.**AngrSurveyorError**

Bases: *AngrError*

**exception** angr.errors.**AngrAnalysisError**

Bases: *AngrError*

**exception** angr.errors.**AngrBladeError**

Bases: *AngrError*

**exception** angr.errors.**AngrBladeSimProcError**

Bases: *AngrBladeError*

**exception** angr.errors.**AngrAnnotatedCFGError**

Bases: *AngrError*

exception angr.errors.**AngrBackwardSlicingError**

Bases: *AngrError*

exception angr.errors.**AngrCallableError**

Bases: *AngrSurveyorError*

exception angr.errors.**AngrCallableMultistateError**

Bases: *AngrCallableError*

exception angr.errors.**AngrSyscallError**

Bases: *AngrError*

exception angr.errors.**AngrSimOSError**

Bases: *AngrError*

exception angr.errors.**AngrAssemblyError**

Bases: *AngrError*

exception angr.errors.**AngrTypeError**

Bases: *AngrError*, TypeError

exception angr.errors.**AngrMissingTypeError**

Bases: *AngrTypeError*

exception angr.errors.**AngrIncongruencyError**

Bases: *AngrAnalysisError*

exception angr.errors.**AngrForwardAnalysisError**

Bases: *AngrError*

exception angr.errors.**AngrSkipJobNotice**

Bases: *AngrForwardAnalysisError*

exception angr.errors.**AngrDelayJobNotice**

Bases: *AngrForwardAnalysisError*

exception angr.errors.**AngrJobMergingFailureNotice**

Bases: *AngrForwardAnalysisError*

exception angr.errors.**AngrJobWideningFailureNotice**

Bases: *AngrForwardAnalysisError*

exception angr.errors.**AngrCFGError**

Bases: *AngrError*

exception angr.errors.**AngrVFGError**

Bases: *AngrError*

exception angr.errors.**AngrVFGRestartAnalysisNotice**

Bases: *AngrVFGError*

exception angr.errors.**AngrDataGraphError**

Bases: *AngrAnalysisError*

exception angr.errors.**AngrDDGError**

Bases: *AngrAnalysisError*

exception angr.errors.**AngrLoopAnalysisError**

Bases: *AngrAnalysisError*

exception angr.errors.**AngrExplorationTechniqueError**

Bases: *AngrError*

exception angr.errors.**AngrExplorerError**

Bases: *AngrExplorationTechniqueError*

exception angr.errors.**AngrDirectorError**

Bases: *AngrExplorationTechniqueError*

exception angr.errors.**AngrTracerError**

Bases: *AngrExplorationTechniqueError*

exception angr.errors.**AngrVariableRecoveryError**

Bases: *AngrAnalysisError*

exception angr.errors.**AngrDBError**

Bases: *AngrError*

exception angr.errors.**AngrCorruptDBError**

Bases: *AngrDBError*

exception angr.errors.**AngrIncompatibleDBError**

Bases: *AngrDBError*

exception angr.errors.**AngrRuntimeDbError**

Bases: *AngrError*

exception angr.errors.**TracerEnvironmentError**

Bases: *AngrError*

exception angr.errors.**SimError**

Bases: Exception

**bbl_addr = None**

**stmt_idx = None**

**ins_addr = None**

**executed_instruction_count = None**

**guard = None**

**record_state**(*state*)

exception angr.errors.**SimStateError**

Bases: *SimError*

exception angr.errors.**SimMergeError**

Bases: *SimStateError*

exception angr.errors.**SimMemoryError**

Bases: *SimStateError*

exception angr.errors.**SimMemoryMissingError**

Bases: *SimMemoryError*

**exception** angr.errors.**SimAbstractMemoryError**

Bases: *SimMemoryError*

**exception** angr.errors.**SimRegionMapError**

Bases: *SimMemoryError*

**exception** angr.errors.**SimMemoryLimitError**

Bases: *SimMemoryError*

**exception** angr.errors.**SimMemoryAddressError**

Bases: *SimMemoryError*

**exception** angr.errors.**SimFastMemoryError**

Bases: *SimMemoryError*

**exception** angr.errors.**SimEventError**

Bases: *SimStateError*

**exception** angr.errors.**SimPosixError**

Bases: *SimStateError*

**exception** angr.errors.**SimFilesystemError**

Bases: *SimError*

**exception** angr.errors.**SimSymbolicFilesystemError**

Bases: *SimFilesystemError*

**exception** angr.errors.**SimFileError**

Bases: *SimMemoryError*, *SimFilesystemError*

**exception** angr.errors.**SimHeapError**

Bases: *SimStateError*

**exception** angr.errors.**SimUnsupportedError**

Bases: *SimError*

**exception** angr.errors.**SimSolverError**

Bases: *SimError*

**exception** angr.errors.**SimSolverModeError**

Bases: *SimSolverError*

**exception** angr.errors.**SimSolverOptionError**

Bases: *SimSolverError*

**exception** angr.errors.**SimValueError**

Bases: *SimSolverError*

**exception** angr.errors.**SimUnsatError**

Bases: *SimValueError*

**exception** angr.errors.**SimOperationError**

Bases: *SimError*

**exception** angr.errors.**UnsupportedIROpError**

Bases: *SimOperationError*, *SimUnsupportedError*

**exception** angr.errors.**SimExpressionError**

Bases: *SimError*

**exception** angr.errors.**UnsupportedIRExprError**
Bases: *SimExpressionError*, *SimUnsupportedError*

**exception** angr.errors.**SimCCallError**
Bases: *SimExpressionError*

**exception** angr.errors.**UnsupportedCCallError**
Bases: *SimCCallError*, *SimUnsupportedError*

**exception** angr.errors.**SimUninitializedAccessError**
Bases: *SimExpressionError*

**exception** angr.errors.**SimStatementError**
Bases: *SimError*

**exception** angr.errors.**UnsupportedIRStmtError**
Bases: *SimStatementError*, *SimUnsupportedError*

**exception** angr.errors.**UnsupportedDirtyError**
Bases: *UnsupportedIRStmtError*, *SimUnsupportedError*

**exception** angr.errors.**SimMissingTempError**
Bases: *SimValueError*, IndexError

**exception** angr.errors.**SimEngineError**
Bases: *SimError*

**exception** angr.errors.**SimIRSBError**
Bases: *SimEngineError*

**exception** angr.errors.**SimTranslationError**
Bases: *SimEngineError*

**exception** angr.errors.**SimProcedureError**
Bases: *SimEngineError*

**exception** angr.errors.**SimProcedureArgumentError**
Bases: *SimProcedureError*

**exception** angr.errors.**SimShadowStackError**
Bases: *SimProcedureError*

**exception** angr.errors.**SimFastPathError**
Bases: *SimEngineError*

**exception** angr.errors.**SimIRSBNoDecodeError**
Bases: *SimIRSBError*

**exception** angr.errors.**AngrUnsupportedSyscallError**
Bases: *AngrSyscallError*, *SimProcedureError*, *SimUnsupportedError*

angr.errors.**UnsupportedSyscallError**
alias of *AngrUnsupportedSyscallError*

**exception** angr.errors.**SimReliftException**
Bases: *SimEngineError*

**exception** angr.errors.**SimSlicerError**
Bases: *SimError*

**exception** angr.errors.**SimActionError**

Bases: *SimError*

**exception** angr.errors.**SimCCError**

Bases: *SimError*

**exception** angr.errors.**SimUCManagerError**

Bases: *SimError*

**exception** angr.errors.**SimUCManagerAllocationError**

Bases: *SimUCManagerError*

**exception** angr.errors.**SimUnicornUnsupport**

Bases: *SimError*

**exception** angr.errors.**SimUnicornError**

Bases: *SimError*

**exception** angr.errors.**SimUnicornSymbolic**

Bases: *SimError*

**exception** angr.errors.**SimEmptyCallStackError**

Bases: *SimError*

**exception** angr.errors.**SimStateOptionsError**

Bases: *SimError*

**exception** angr.errors.**SimException**

Bases: *SimError*

**exception** angr.errors.**SimSegfaultException**

Bases: *SimException*, *SimMemoryError*

angr.errors.**SimSegfaultError**

alias of *SimSegfaultException*

**exception** angr.errors.**SimZeroDivisionException**

Bases: *SimException*, *SimOperationError*

**exception** angr.errors.**AngrNoPluginError**

Bases: *AngrError*

**exception** angr.errors.**SimConcreteMemoryError**

Bases: *AngrError*

**exception** angr.errors.**SimConcreteRegisterError**

Bases: *AngrError*

**exception** angr.errors.**SimConcreteBreakpointError**

Bases: *AngrError*

**exception** angr.errors.**AngrDecompilationError**

Bases: *AngrError*

**exception** angr.errors.**AngrDecompilationComplexityError**

Bases: *AngrDecompilationError*

Raised when a function exceeds one of the decompiler's complexity limits. The decompiler gives up before starting the expensive work instead of running for an unbounded amount of time.

**Variables**

- **limit_name** – Name of the decompilation option that sets the limit, e.g. "max_function_blocks".
- **limit** – The configured limit.
- **actual** – The measured size that exceeded the limit.
- **func_addr** – Address of the offending function, if known.
- **unit** – What `actual` counts, e.g. "basic blocks".

**exception** angr.errors.**UnsupportedNodeTypeError**

Bases: *AngrError*, NotImplementedError

**exception** angr.errors.**AngrAIError**

Bases: *AngrError*

## 10.17 angr.exploration_techniques

**class** angr.exploration_techniques.**DFS**

Bases: *ExplorationTechnique*

Depth-first search.

Will only keep one path active at a time, any others will be stashed in the 'deferred' stash. When we run out of active paths to step, we take the longest one from deferred and continue.

**__init__**(*deferred_stash='deferred'*)

**class** angr.exploration_techniques.**Bucketizer**

Bases: *ExplorationTechnique*

Loop bucketization: Pick log(n) paths out of n possible paths, and stash (or drop) everything else.

**class** angr.exploration_techniques.**CallFunctionGoal**

Bases: *BaseGoal*

A goal that prioritizes states reaching certain function, and optionally with specific arguments. Note that constraints on arguments (and on function address as well) have to be identifiable on an accurate CFG. For example, you may have a CallFunctionGoal saying "call printf with the first argument being 'Hello, world'", and CFGEmulated must be able to figure our the first argument to printf is in fact "Hello, world", not some symbolic strings that will be constrained to "Hello, world" during symbolic execution (or simulation, however you put it).

**REQUIRE_CFG_STATES = True**

**__init__**(*function*, *arguments*)

**check**(*cfg*, *state*, *peek_blocks*)

Check if the specified function will be reached with certain arguments.

**Parameters**

- **cfg**
- **state**
- **peek_blocks**

**Returns**

**check_state**(*state*)

Check if the specific function is reached with certain arguments

**Parameters**
**state** (*angr.SimState*) – The state to check

**Returns**
True if the function is reached with certain arguments, False otherwise.

**Return type**
bool

**class** angr.exploration_techniques.**Director**

Bases: *ExplorationTechnique*

An exploration technique for directed symbolic execution.

A control flow graph (using CFGEmulated) is built and refined during symbolic execution. Each time the execution reaches a block that is outside of the CFG, the CFG recovery will be triggered with that state, with a maximum recovery depth (100 by default). If we see a basic block during state stepping that is not yet in the control flow graph, we go back to control flow graph recovery and "peek" more blocks forward.

When stepping a simulation manager, all states are categorized into three different categories:

- Might reach the destination within the peek depth. Those states are prioritized.
- Will not reach the destination within the peek depth. Those states are de-prioritized. However, there is a little chance for those states to be explored as well in order to prevent over-fitting.

**__init__**(*peek_blocks=100*, *peek_functions=5*, *goals=None*, *cfg_keep_states=False*, *goal_satisfied_callback=None*, *num_fallback_states=5*)

Constructor.

**step**(*simgr*, *stash='active'*, ***kwargs*)

**Parameters**

- **simgr**
- **stash**
- **kwargs**

**Returns**

**add_goal**(*goal*)

Add a goal.

**Parameters**
**goal** (BaseGoal) – The goal to add.

**Returns**
None

**class** angr.exploration_techniques.**DrillerCore**

Bases: *ExplorationTechnique*

An exploration technique that symbolically follows an input looking for new state transitions.

It has to be used with Tracer exploration technique. Results are put in 'diverted' stash.

**__init__**(*trace*, *fuzz_bitmap=None*)

:param trace : The basic block trace. :type fuzz_bitmap: :param fuzz_bitmap: AFL's bitmap of state transitions. Defaults to saying every transition is worth satisfying.

**class** angr.exploration_techniques.**ExecuteAddressGoal**

Bases: *BaseGoal*

A goal that prioritizes states reaching (or are likely to reach) certain address in some specific steps.

**__init__**(*addr*)

**check**(*cfg*, *state*, *peek_blocks*)

Check if the specified address will be executed

**Parameters**

- **cfg**
- **state**
- **peek_blocks** (*int*)

**Returns**

**Return type**
bool

**check_state**(*state*)

Check if the current address is the target address.

**Parameters**
**state** (*angr.SimState*) – The state to check.

**Returns**
True if the current address is the target address, False otherwise.

**Return type**
bool

**class** angr.exploration_techniques.**ExplorationTechnique**

Bases: object

An ExplorationTechnique is a set of hooks for a simulation manager that assists in the implementation of new techniques in symbolic exploration.

Any number of these methods may be overridden by a subclass. To use an exploration technique, call `simgr.use_technique` with an *instance* of the technique.

**__init__**()

**setup**(*simgr*)

Perform any initialization on this manager you might need to do.

**Parameters**
**simgr** (*angr.SimulationManager*) – The simulation manager to which you have just been added

**step**(*simgr*, *stash='active'*, ***kwargs*)

Hook the process of stepping a stash forward. Should call `simgr.step(stash, **kwargs)` in order to do the actual processing.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **stash** (*str*)

**filter**(*simgr*, *state*, ***kwargs*)

Perform filtering on which stash a state should be inserted into.

If the state should be filtered, return the name of the stash to move the state to. If you want to modify the state before filtering it, return a tuple of the stash to move the state to and the modified state. To defer to the original categorization procedure, return the result of `simgr.filter(state, **kwargs)`

If the user provided a `filter_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**selector**(*simgr*, *state*, ***kwargs*)

Determine if a state should participate in the current round of stepping. Return True if the state should be stepped, and False if the state should not be stepped. To defer to the original selection procedure, return the result of `simgr.selector(state, **kwargs)`.

If the user provided a `selector_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**step_state**(*simgr*, *state*, ***kwargs*)

Determine the categorization of state successors into stashes. The result should be a dict mapping stash names to the list of successor states that fall into that stash, or None as a stash name to use the original stash name.

If you would like to directly work with a *SimSuccessors* object, you can obtain it with `simgr.successors(state, **kwargs)`. This is not recommended, as it denies other hooks the opportunity to look at the successors. Therefore, the usual technique is to call `simgr.step_state(state, **kwargs)` and then mutate the returned dict before returning it yourself.

..note:: This takes precedence over the *filter* hook - *filter* is only applied to states returned from here in the None stash.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**successors**(*simgr*, *state*, ***kwargs*)

Perform the process of stepping a state forward, returning a SimSuccessors object.

To defer to the original succession procedure, return the result of `simgr.successors(state, **kwargs)`. Be careful about not calling this method (e.g. calling `project.factory.successors` manually) as it denies other hooks the opportunity to instrument the step. Instead, you can mutate the kwargs for the step before calling the original, and mutate the result before returning it yourself.

If the user provided a `successor_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)

- **state** (*angr.SimState*)

**complete**(*simgr*)

Return whether or not this manager has reached a "completed" state, i.e. `SimulationManager.run()` should halt.

This is the one hook which is *not* subject to the nesting rules of hooks. You should *not* call `simgr.complete`, you should make your own decision and return True or False. Each of the techniques' completion checkers will be called and the final result will be compted with `simgr.completion_mode`.

**Parameters**
**simgr** (*angr.SimulationManager*)

**class** angr.exploration_techniques.**Explorer**

Bases: *ExplorationTechnique*

Search for up to "num_find" paths that satisfy condition "find", avoiding condition "avoid". Stashes found paths into "find_stash' and avoided paths into "avoid_stash".

The "find" and "avoid" parameters may be any of:

- An address to find
- A set or list of addresses to find
- A function that takes a path and returns whether or not it matches.

If an angr CFG is passed in as the "cfg" parameter and "find" is either a number or a list or a set, then any paths which cannot possibly reach a success state without going through a failure state will be preemptively avoided.

If either the "find" or "avoid" parameter is a function returning a boolean, and a path triggers both conditions, it will be added to the find stash, unless "avoid_priority" is set to True.

**__init__**(*find=None*, *avoid=None*, *find_stash='found'*, *avoid_stash='avoid'*, *cfg=None*, *num_find=1*, *avoid_priority=False*)

**class** angr.exploration_techniques.**LengthLimiter**

Bases: *ExplorationTechnique*

Length limiter on paths.

**__init__**(*max_length*, *drop=False*)

**class** angr.exploration_techniques.**LocalLoopSeer**

Bases: *ExplorationTechnique*

LocalLoopSeer monitors exploration and maintains all loop-related data without relying on a control flow graph.

**__init__**(*bound=None*, *bound_reached=None*, *discard_stash='spinning'*)

**Parameters**

- **bound** – Limit the number of iterations a loop may be executed.
- **bound_reached** – If provided, should be a function that takes the LoopSeer and the succ_state. Will be called when loop execution reach the given bound. Default to moving states that exceed the loop limit to a discard stash.
- **discard_stash** – Name of the stash containing states exceeding the loop limit.

class angr.exploration_techniques.**LoopSeer**

Bases: *ExplorationTechnique*

This exploration technique monitors exploration and maintains all loop-related data (well, currently it is just the loop trip counts, but feel free to add something else).

**__init__**(*cfg=None*, *functions=None*, *loops=None*, *use_header=False*, *bound=None*, *bound_reached=None*, *discard_stash='spinning'*, *limit_concrete_loops=True*)

**Parameters**

- **cfg** – Normalized CFG is required.
- **functions** – Function(s) containing the loop(s) to be analyzed.
- **loops** – Specific group of Loop(s) to be analyzed, if this is None we run the LoopFinder analysis.
- **use_header** – Whether to use header based trip counter to compare with the bound limit.
- **bound** – Limit the number of iterations a loop may be executed.
- **bound_reached** – If provided, should be a function that takes the LoopSeer and the succ_state. Will be called when loop execution reach the given bound. Default to moving states that exceed the loop limit to a discard stash.
- **discard_stash** – Name of the stash containing states exceeding the loop limit.
- **limit_concrete_loops** – If False, do not limit a loop back-edge if it is the only successor (Defaults to True to maintain the original behavior)

class angr.exploration_techniques.**ManualMergepoint**

Bases: *ExplorationTechnique*

**__init__**(*address*, *wait_counter=10*, *prune=True*)

**mark_nofilter**(*simgr*, *stash*)

**mark_okfilter**(*simgr*, *stash*)

class angr.exploration_techniques.**MemoryWatcher**

Bases: *ExplorationTechnique*

Memory Watcher

**Parameters**

- **min_memory** (*int*, *optional*) – Minimum amount of free memory in MB before stopping execution (default: 95% memory use)
- **memory_stash** (*str*, *optional*) – What to call the low memory stash (default: 'lowmem')

At each step, keep an eye on how much memory is left on the system. Stash off states to effectively stop execution if we're below a given threshold.

**__init__**(*min_memory=512*, *memory_stash='lowmem'*)

class angr.exploration_techniques.**Oppologist**

Bases: *ExplorationTechnique*

The Oppologist is an exploration technique that forces uncooperative code through qemu.

__init__()

class angr.exploration_techniques.**Slicecutor**

Bases: *ExplorationTechnique*

The Slicecutor is an exploration that executes provided code slices.

__init__(*annotated_cfg*, *force_taking_exit=False*, *force_sat=False*)

All parameters except *annotated_cfg* are optional.

**Parameters**

- **annotated_cfg** – The AnnotatedCFG that provides the code slice.
- **force_taking_exit** – Set to True if you want to create a successor based on our slice in case of unconstrained successors.
- **force_sat** (bool) – If a branch specified by the slice is unsatisfiable, set this option to True if you want to force it to be satisfiable and be taken anyway.

class angr.exploration_techniques.**Spiller**

Bases: *ExplorationTechnique*

Automatically spill states out. It can spill out states to a different stash, spill them out to ANA, or first do the former and then (after enough states) the latter.

__init__(*src_stash='active'*, *min=5*, *max=10*, *staging_stash='spill_stage'*, *staging_min=10*, *staging_max=20*, *pickle_callback=None*, *unpickle_callback=None*, *post_pickle_callback=None*, *priority_key=None*, *vault=None*, *states_collection=None*)

Initializes the spiller.

**Parameters**

- **max** – the number of states that are *not* spilled
- **src_stash** – the stash from which to spill states (default: active)
- **staging_stash** – the stash *to* which to spill states (default: "spill_stage")
- **staging_max** – the number of states that can be in the staging stash before things get spilled to ANA (default: None. If staging_stash is set, then this means unlimited, and ANA will not be used).
- **priority_key** – a function that takes a state and returns its numerical priority (MAX_INT is lowest priority). By default, self.state_priority will be used, which prioritizes by object ID.
- **vault** – an angr.Vault object to handle storing and loading of states. If not provided, an angr.vaults.VaultShelf will be created with a temporary file.

static **state_priority**(*state*)

class angr.exploration_techniques.**StochasticSearch**

Bases: *ExplorationTechnique*

Stochastic Search.

Will only keep one path active at a time, any others will be discarded. Before each pass through, weights are randomly assigned to each basic block. These weights form a probability distribution for determining which state remains after splits. When we run out of active paths to step, we start again from the start state.

__init__(*start_state*, *restart_prob=0.0001*)

**Parameters**

- **start_state** – The initial state from which exploration stems.
- **restart_prob** – The probability of randomly restarting the search (default 0.0001).

class angr.exploration_techniques.**StubStasher**

Bases: *ExplorationTechnique*

Stash states that reach a stub SimProcedure.

static **post_filter**(*state*)

class angr.exploration_techniques.**Suggestions**

Bases: *ExplorationTechnique*

An exploration technique which analyzes failure cases and logs suggestions for how to mitigate them in future analyses.

__init__()

static **report**(*state*, *event*)

class angr.exploration_techniques.**TechniqueBuilder**

Bases: *ExplorationTechnique*

This meta technique could be used to hook a couple of simulation manager methods without actually creating a new exploration technique, for example:

class SomeComplexAnalysis(Analysis):

**def do_something():**
simgr = self.project.factory.simulation_manager() simgr.use_tech(ProxyTechnique(step_state=self._step_state))
simgr.run()

**def _step_state(self, state):**
# Do stuff! pass

In the above example, the _step_state method can access all the necessary stuff, hidden in the analysis instance, without passing that instance to a one-shot-styled exploration technique.

__init__(*setup=None*, *step_state=None*, *step=None*, *successors=None*, *filter=None*, *selector=None*, *complete=None*)

class angr.exploration_techniques.**Threading**

Bases: *ExplorationTechnique*

Enable multithreading.

This is only useful in paths where a lot of time is taken inside z3, doing constraint solving. This is because of python's GIL, which says that only one thread at a time may be executing python code.

__init__(*threads=8*, *local_stash='thread_local'*)

inner_step(*state*, *simgr*, ***kwargs*)

class angr.exploration_techniques.**Timeout**

Bases: *ExplorationTechnique*

Timeout exploration technique that stops an active exploration if the run time exceeds a predefined timeout

__init__(*timeout=None*)

class angr.exploration_techniques.**Tracer**

Bases: *ExplorationTechnique*

An exploration technique that follows an angr path with a concrete input. The tracing result is the state at the last address of the trace, which can be found in the 'traced' stash.

If the given concrete input makes the program crash, you should provide crash_addr, and the crashing state will be found in the 'crashed' stash.

**Parameters**

- **trace** (list[int]) – The basic block trace.
- **resiliency** – Should we continue to step forward even if qemu and angr disagree?
- **keep_predecessors** – Number of states before the final state we should log.
- **crash_addr** – If the trace resulted in a crash, provide the crashing instruction pointer here, and the 'crashed' stash will be populated with the crashing state.
- **syscall_data** – Data related to various syscalls recorded by tracer for replaying
- **copy_states** – Whether COPY_STATES should be enabled for the tracing state. It is off by default because most tracing workloads benefit greatly from not performing copying. You want to enable it if you want to see the missed states. It will be re-added for the last 2% of the trace in order to set the predecessors list correctly. If you turn this on you may want to enable the LAZY_SOLVES option.
- **mode** – Tracing mode.
- **aslr** – Whether there are aslr slides. if not, tracer uses trace address as state address.
- **follow_unsat** – Whether unsatisfiable states should be treated as potential successors or not.

**Variables**

**predecessors** (list[SimState]) – A list of states in the history before the final state.

__init__(*trace*, *resiliency=False*, *keep_predecessors=1*, *crash_addr=None*, *syscall_data=None*, *copy_states=False*, *fast_forward_to_entry=True*, *mode='strict'*, *aslr=True*, *follow_unsat=False*)

**Parameters**

**trace** (*list[int]*)

set_fd_data(*fd_data*)

Set concrete bytes of various fds read by the program

**Parameters**

**fd_data** (*dict[int, bytes]*)

classmethod **crash_windup**(*state*, *crash_addr*)

class angr.exploration_techniques.**UniqueSearch**

Bases: *ExplorationTechnique*

Unique Search.

Will only keep one path active at a time, any others will be deferred. The state that is explored depends on how unique it is relative to the other deferred states. A path's uniqueness is determined by its average similarity between the other (deferred) paths. Similarity is calculated based on the supplied *similarity_func*, which by default is: The (L2) distance between the counts of the state addresses in the history of the path.

**__init__**(*similarity_func=None*, *deferred_stash='deferred'*)

**Parameters**

- **similarity_func** – How to calculate similarity between two states.
- **deferred_stash** – Where to store the deferred states.

**static similarity**(*state_a*, *state_b*)

The (L2) distance between the counts of the state addresses in the history of the path. :type state_a: :param state_a: The first state to compare :type state_b: :param state_b: The second state to compare

**static sequence_matcher_similarity**(*state_a*, *state_b*)

The *difflib.SequenceMatcher* ratio between the state addresses in the history of the path. :type state_a: :param state_a: The first state to compare :type state_b: :param state_b: The second state to compare

**class** angr.exploration_techniques.**Veritesting**

Bases: *ExplorationTechnique*

Enable veritesting. This technique, described in a paper[1] from CMU, attempts to address the problem of state explosions in loops by performing smart merging.

[1] https://users.ece.cmu.edu/~aavgerin/papers/veritesting-icse-2014.pdf

**__init__**(***options*)

## Submodules

| |
|---|
| *base* |
| *bucketizer* |
| *common* |
| *dfs* |
| *director* |
| *driller_core* |
| *explorer* |
| *lengthlimiter* |
| *local_loop_seer* |
| *loop_seer* |
| *manual_mergepoint* |
| *memory_watcher* |
| *oppologist* |
| *slicecutor* |
| *spiller* |
| *spiller_db* |
| *stochastic* |
| *stub_stasher* |
| *suggestions* |
| *tech_builder* |
| *threading* |
| *timeout* |
| *tracer* |
| *unique* |
| *veritesting* |

### 10.17.1 angr.exploration_techniques.base

**class** angr.exploration_techniques.base.**ExplorationTechnique**

Bases: object

An ExplorationTechnique is a set of hooks for a simulation manager that assists in the implementation of new techniques in symbolic exploration.

Any number of these methods may be overridden by a subclass. To use an exploration technique, call `simgr.use_technique` with an *instance* of the technique.

**__init__**()

**setup**(*simgr*)

Perform any initialization on this manager you might need to do.

**Parameters**

**simgr** (*angr.SimulationManager*) – The simulation manager to which you have just been added

**step**(*simgr*, *stash='active'*, ***kwargs*)

Hook the process of stepping a stash forward. Should call `simgr.step(stash, **kwargs)` in order to do the actual processing.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **stash** (*str*)

**filter**(*simgr*, *state*, ***kwargs*)

Perform filtering on which stash a state should be inserted into.

If the state should be filtered, return the name of the stash to move the state to. If you want to modify the state before filtering it, return a tuple of the stash to move the state to and the modified state. To defer to the original categorization procedure, return the result of `simgr.filter(state, **kwargs)`

If the user provided a `filter_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**selector**(*simgr*, *state*, ***kwargs*)

Determine if a state should participate in the current round of stepping. Return True if the state should be stepped, and False if the state should not be stepped. To defer to the original selection procedure, return the result of `simgr.selector(state, **kwargs)`.

If the user provided a `selector_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**step_state**(*simgr*, *state*, ***kwargs*)

Determine the categorization of state successors into stashes. The result should be a dict mapping stash names to the list of successor states that fall into that stash, or None as a stash name to use the original stash name.

If you would like to directly work with a *SimSuccessors* object, you can obtain it with `simgr.successors(state, **kwargs)`. This is not recommended, as it denies other hooks the opportunity to look at the successors. Therefore, the usual technique is to call `simgr.step_state(state, **kwargs)` and then mutate the returned dict before returning it yourself.

..note:: This takes precedence over the *filter* hook - *filter* is only applied to states returned from here in the None stash.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**successors**(*simgr*, *state*, ***kwargs*)

Perform the process of stepping a state forward, returning a SimSuccessors object.

To defer to the original succession procedure, return the result of `simgr.successors(state, **kwargs)`. Be careful about not calling this method (e.g. calling `project.factory.successors` manually) as it denies other hooks the opportunity to instrument the step. Instead, you can mutate the kwargs for the step before calling the original, and mutate the result before returning it yourself.

If the user provided a `successor_func` in their step or run command, it will appear here.

**Parameters**

- **simgr** (*angr.SimulationManager*)
- **state** (*angr.SimState*)

**complete**(*simgr*)

Return whether or not this manager has reached a "completed" state, i.e. `SimulationManager.run()` should halt.

This is the one hook which is *not* subject to the nesting rules of hooks. You should *not* call `simgr.complete`, you should make your own decision and return True or False. Each of the techniques' completion checkers will be called and the final result will be compted with `simgr.completion_mode`.

**Parameters**

**simgr** (*angr.SimulationManager*)

### 10.17.2 angr.exploration_techniques.bucketizer

**class** angr.exploration_techniques.bucketizer.**Bucketizer**

Bases: *ExplorationTechnique*

Loop bucketization: Pick log(n) paths out of n possible paths, and stash (or drop) everything else.

### 10.17.3 angr.exploration_techniques.common

angr.exploration_techniques.common.**condition_to_lambda**(*condition*, *default=False*)

Translates an integer, set, list or function into a lambda that checks if state's current basic block matches some condition.

**Parameters**

- **condition** – An integer, set, list or lambda to convert to a lambda.
- **default** – The default return value of the lambda (in case condition is None). Default: false.

**Returns**
A tuple of two items: a lambda that takes a state and returns the set of addresses that it matched from the condition, and a set that contains the normalized set of addresses to stop at, or None if no addresses were provided statically.

### 10.17.4 angr.exploration_techniques.dfs

**class** angr.exploration_techniques.dfs.**DFS**

Bases: *ExplorationTechnique*

Depth-first search.

Will only keep one path active at a time, any others will be stashed in the 'deferred' stash. When we run out of active paths to step, we take the longest one from deferred and continue.

**__init__**(*deferred_stash='deferred'*)

### 10.17.5 angr.exploration_techniques.director

**class** angr.exploration_techniques.director.**BaseGoal**

Bases: object

**REQUIRE_CFG_STATES = False**

**__init__**(*sort*)

**check**(*cfg*, *state*, *peek_blocks*)

**Parameters**

- **cfg** (*angr.analyses.CFGEmulated*) – An instance of CFGEmulated.
- **state** (*angr.SimState*) – The state to check.
- **peek_blocks** (*int*) – Number of blocks to peek ahead from the current point.

**Returns**
True if we can determine that this condition is definitely satisfiable if the path is taken, False otherwise.

**Return type**
bool

**check_state**(*state*)

Check if the current state satisfies the goal.

**Parameters**
**state** (*angr.SimState*) – The state to check.

**Returns**
True if it satisfies the goal, False otherwise.

**Return type**
bool

**class** angr.exploration_techniques.director.**ExecuteAddressGoal**

Bases: *BaseGoal*

A goal that prioritizes states reaching (or are likely to reach) certain address in some specific steps.

**__init__**(*addr*)

**check**(*cfg*, *state*, *peek_blocks*)

Check if the specified address will be executed

**Parameters**

- **cfg**
- **state**
- **peek_blocks** (*int*)

**Returns**

**Return type**
bool

**check_state**(*state*)

Check if the current address is the target address.

**Parameters**
**state** (*angr.SimState*) – The state to check.

**Returns**
True if the current address is the target address, False otherwise.

**Return type**
bool

**class** angr.exploration_techniques.director.**CallFunctionGoal**

Bases: *BaseGoal*

A goal that prioritizes states reaching certain function, and optionally with specific arguments. Note that constraints on arguments (and on function address as well) have to be identifiable on an accurate CFG. For example, you may have a CallFunctionGoal saying "call printf with the first argument being 'Hello, world'", and CFGEmulated must be able to figure our the first argument to printf is in fact "Hello, world", not some symbolic strings that will be constrained to "Hello, world" during symbolic execution (or simulation, however you put it).

**REQUIRE_CFG_STATES = True**

**__init__**(*function*, *arguments*)

**check**(*cfg*, *state*, *peek_blocks*)

Check if the specified function will be reached with certain arguments.

**Parameters**

- **cfg**
- **state**
- **peek_blocks**

**Returns**

**check_state**(*state*)

Check if the specific function is reached with certain arguments

**Parameters**
**state** (*angr.SimState*) – The state to check

**Returns**
True if the function is reached with certain arguments, False otherwise.

**Return type**
bool

**class** angr.exploration_techniques.director.**Director**

Bases: *ExplorationTechnique*

An exploration technique for directed symbolic execution.

A control flow graph (using CFGEmulated) is built and refined during symbolic execution. Each time the execution reaches a block that is outside of the CFG, the CFG recovery will be triggered with that state, with a maximum recovery depth (100 by default). If we see a basic block during state stepping that is not yet in the control flow graph, we go back to control flow graph recovery and "peek" more blocks forward.

When stepping a simulation manager, all states are categorized into three different categories:

- Might reach the destination within the peek depth. Those states are prioritized.
- Will not reach the destination within the peek depth. Those states are de-prioritized. However, there is a little chance for those states to be explored as well in order to prevent over-fitting.

**__init__**(*peek_blocks=100*, *peek_functions=5*, *goals=None*, *cfg_keep_states=False*, *goal_satisfied_callback=None*, *num_fallback_states=5*)

Constructor.

**step**(*simgr*, *stash='active'*, ***kwargs*)

**Parameters**

- **simgr**
- **stash**
- **kwargs**

**Returns**

**add_goal**(*goal*)

Add a goal.

**Parameters**
**goal** (BaseGoal) – The goal to add.

**Returns**
None

### 10.17.6 angr.exploration_techniques.driller_core

**class** angr.exploration_techniques.driller_core.**DrillerCore**

Bases: *ExplorationTechnique*

An exploration technique that symbolically follows an input looking for new state transitions.

It has to be used with Tracer exploration technique. Results are put in 'diverted' stash.

**__init__**(*trace*, *fuzz_bitmap=None*)

:param trace : The basic block trace. :type fuzz_bitmap: :param fuzz_bitmap: AFL's bitmap of state transitions. Defaults to saying every transition is worth satisfying.

### 10.17.7 angr.exploration_techniques.explorer

**class** angr.exploration_techniques.explorer.**Explorer**

Bases: *ExplorationTechnique*

Search for up to "num_find" paths that satisfy condition "find", avoiding condition "avoid". Stashes found paths into "find_stash' and avoided paths into "avoid_stash".

The "find" and "avoid" parameters may be any of:

- An address to find
- A set or list of addresses to find
- A function that takes a path and returns whether or not it matches.

If an angr CFG is passed in as the "cfg" parameter and "find" is either a number or a list or a set, then any paths which cannot possibly reach a success state without going through a failure state will be preemptively avoided.

If either the "find" or "avoid" parameter is a function returning a boolean, and a path triggers both conditions, it will be added to the find stash, unless "avoid_priority" is set to True.

**__init__**(*find=None*, *avoid=None*, *find_stash='found'*, *avoid_stash='avoid'*, *cfg=None*, *num_find=1*, *avoid_priority=False*)

### 10.17.8 angr.exploration_techniques.lengthlimiter

**class** angr.exploration_techniques.lengthlimiter.**LengthLimiter**

Bases: *ExplorationTechnique*

Length limiter on paths.

**__init__**(*max_length*, *drop=False*)

### 10.17.9 angr.exploration_techniques.local_loop_seer

**class** angr.exploration_techniques.local_loop_seer.**LocalLoopSeer**

Bases: *ExplorationTechnique*

LocalLoopSeer monitors exploration and maintains all loop-related data without relying on a control flow graph.

**__init__**(*bound=None*, *bound_reached=None*, *discard_stash='spinning'*)

**Parameters**

- **bound** – Limit the number of iterations a loop may be executed.
- **bound_reached** – If provided, should be a function that takes the LoopSeer and the succ_state. Will be called when loop execution reach the given bound. Default to moving states that exceed the loop limit to a discard stash.
- **discard_stash** – Name of the stash containing states exceeding the loop limit.

### 10.17.10 angr.exploration_techniques.loop_seer

**class** angr.exploration_techniques.loop_seer.**LoopSeer**

Bases: *ExplorationTechnique*

This exploration technique monitors exploration and maintains all loop-related data (well, currently it is just the loop trip counts, but feel free to add something else).

__init__(*cfg=None*, *functions=None*, *loops=None*, *use_header=False*, *bound=None*, *bound_reached=None*, *discard_stash='spinning'*, *limit_concrete_loops=True*)

**Parameters**

- **cfg** – Normalized CFG is required.
- **functions** – Function(s) containing the loop(s) to be analyzed.
- **loops** – Specific group of Loop(s) to be analyzed, if this is None we run the LoopFinder analysis.
- **use_header** – Whether to use header based trip counter to compare with the bound limit.
- **bound** – Limit the number of iterations a loop may be executed.
- **bound_reached** – If provided, should be a function that takes the LoopSeer and the succ_state. Will be called when loop execution reach the given bound. Default to moving states that exceed the loop limit to a discard stash.
- **discard_stash** – Name of the stash containing states exceeding the loop limit.
- **limit_concrete_loops** – If False, do not limit a loop back-edge if it is the only successor (Defaults to True to maintain the original behavior)

### 10.17.11 angr.exploration_techniques.manual_mergepoint

class angr.exploration_techniques.manual_mergepoint.**ManualMergepoint**

Bases: *ExplorationTechnique*

__init__(*address*, *wait_counter=10*, *prune=True*)

mark_nofilter(*simgr*, *stash*)

mark_okfilter(*simgr*, *stash*)

### 10.17.12 angr.exploration_techniques.memory_watcher

class angr.exploration_techniques.memory_watcher.**MemoryWatcher**

Bases: *ExplorationTechnique*

Memory Watcher

**Parameters**

- **min_memory** (*int*, *optional*) – Minimum amount of free memory in MB before stopping execution (default: 95% memory use)
- **memory_stash** (*str*, *optional*) – What to call the low memory stash (default: 'lowmem')

At each step, keep an eye on how much memory is left on the system. Stash off states to effectively stop execution if we're below a given threshold.

__init__(*min_memory=512*, *memory_stash='lowmem'*)

### 10.17.13 angr.exploration_techniques.oppologist

class angr.exploration_techniques.oppologist.**Oppologist**

Bases: *ExplorationTechnique*

The Oppologist is an exploration technique that forces uncooperative code through qemu.

**__init__**()

### 10.17.14 angr.exploration_techniques.slicecutor

class angr.exploration_techniques.slicecutor.**Slicecutor**

Bases: *ExplorationTechnique*

The Slicecutor is an exploration that executes provided code slices.

**__init__**(*annotated_cfg*, *force_taking_exit=False*, *force_sat=False*)

All parameters except *annotated_cfg* are optional.

**Parameters**

- **annotated_cfg** – The AnnotatedCFG that provides the code slice.
- **force_taking_exit** – Set to True if you want to create a successor based on our slice in case of unconstrained successors.
- **force_sat** (bool) – If a branch specified by the slice is unsatisfiable, set this option to True if you want to force it to be satisfiable and be taken anyway.

### 10.17.15 angr.exploration_techniques.spiller

class angr.exploration_techniques.spiller.**PickledStatesBase**

Bases: object

The base class of pickled states

**sort**()

Sort pickled states.

**add**(*prio*, *sid*)

Add a newly pickled state.

**Parameters**

- **prio** (*int*) – Priority of the state.
- **sid** (*str*) – Persistent ID of the state.

**Returns**

None

**pop_n**(*n*)

Pop the top N states.

**Parameters**

**n** (*int*) – Number of states to take.

**Returns**

A list of states.

class angr.exploration_techniques.spiller.**PickledStatesList**

Bases: *PickledStatesBase*

List-backed pickled state storage.

__init__()

class angr.exploration_techniques.spiller.PickledStatesDb

Bases: PickledStatesBase

Database-backed pickled state storage.

__init__(*db_str='sqlite:///:memory:'*)

get_recent_n(*n*, *stash='spilled'*)

count()

class angr.exploration_techniques.spiller.Spiller

Bases: ExplorationTechnique

Automatically spill states out. It can spill out states to a different stash, spill them out to ANA, or first do the former and then (after enough states) the latter.

__init__(*src_stash='active'*, *min=5*, *max=10*, *staging_stash='spill_stage'*, *staging_min=10*, *staging_max=20*, *pickle_callback=None*, *unpickle_callback=None*, *post_pickle_callback=None*, *priority_key=None*, *vault=None*, *states_collection=None*)

Initializes the spiller.

**Parameters**

- **max** – the number of states that are *not* spilled
- **src_stash** – the stash from which to spill states (default: active)
- **staging_stash** – the stash *to* which to spill states (default: "spill_stage")
- **staging_max** – the number of states that can be in the staging stash before things get spilled to ANA (default: None. If staging_stash is set, then this means unlimited, and ANA will not be used).
- **priority_key** – a function that takes a state and returns its numerical priority (MAX_INT is lowest priority). By default, self.state_priority will be used, which prioritizes by object ID.
- **vault** – an angr.Vault object to handle storing and loading of states. If not provided, an angr.vaults.VaultShelf will be created with a temporary file.

static state_priority(*state*)

### 10.17.16 angr.exploration_techniques.spiller_db

class angr.exploration_techniques.spiller_db.PickledState

Bases: Base

id

priority

taken

stash

timestamp

__init__(**kwargs)

A simple constructor that allows initialization from kwargs.

Sets attributes on the constructed instance using the names and values in kwargs.

Only keys that are present as attributes of the instance's class are allowed. These could be, for example, any mapped columns or relationships.

### 10.17.17 angr.exploration_techniques.stochastic

class angr.exploration_techniques.stochastic.StochasticSearch

Bases: ExplorationTechnique

Stochastic Search.

Will only keep one path active at a time, any others will be discarded. Before each pass through, weights are randomly assigned to each basic block. These weights form a probability distribution for determining which state remains after splits. When we run out of active paths to step, we start again from the start state.

__init__(*start_state*, *restart_prob=0.0001*)

**Parameters**

- **start_state** – The initial state from which exploration stems.
- **restart_prob** – The probability of randomly restarting the search (default 0.0001).

### 10.17.18 angr.exploration_techniques.stub_stasher

class angr.exploration_techniques.stub_stasher.StubStasher

Bases: ExplorationTechnique

Stash states that reach a stub SimProcedure.

static post_filter(*state*)

### 10.17.19 angr.exploration_techniques.suggestions

angr.exploration_techniques.suggestions.ast_weight(*ast*, *memo=None*)

class angr.exploration_techniques.suggestions.Suggestions

Bases: ExplorationTechnique

An exploration technique which analyzes failure cases and logs suggestions for how to mitigate them in future analyses.

__init__()

static report(*state*, *event*)

### 10.17.20 angr.exploration_techniques.tech_builder

class angr.exploration_techniques.tech_builder.TechniqueBuilder

Bases: ExplorationTechnique

This meta technique could be used to hook a couple of simulation manager methods without actually creating a new exploration technique, for example:

class SomeComplexAnalysis(Analysis):

**def do_something():**
simgr = self.project.factory.simulation_manager() simgr.use_tech(ProxyTechnique(step_state=self._step_state)) simgr.run()

**def _step_state(self, state):**
# Do stuff! pass

In the above example, the _step_state method can access all the necessary stuff, hidden in the analysis instance, without passing that instance to a one-shot-styled exploration technique.

**__init__**(*setup=None*, *step_state=None*, *step=None*, *successors=None*, *filter=None*, *selector=None*, *complete=None*)

### 10.17.21 angr.exploration_techniques.threading

**class** angr.exploration_techniques.threading.**Threading**

Bases: *ExplorationTechnique*

Enable multithreading.

This is only useful in paths where a lot of time is taken inside z3, doing constraint solving. This is because of python's GIL, which says that only one thread at a time may be executing python code.

**__init__**(*threads=8*, *local_stash='thread_local'*)

**inner_step**(*state*, *simgr*, ***kwargs*)

### 10.17.22 angr.exploration_techniques.timeout

**class** angr.exploration_techniques.timeout.**Timeout**

Bases: *ExplorationTechnique*

Timeout exploration technique that stops an active exploration if the run time exceeds a predefined timeout

**__init__**(*timeout=None*)

### 10.17.23 angr.exploration_techniques.tracer

**class** angr.exploration_techniques.tracer.**TracingMode**

Bases: object

**Variables**

- **Strict** – Strict mode, the default mode, where an exception is raised immediately if tracer's path deviates from the provided trace.
- **Permissive** – Permissive mode, where tracer attempts to force the path back to the provided trace when a deviation happens. This does not always work, especially when the cause of deviation is related to input that will later be used in exploit generation. But, it might work magically sometimes.
- **CatchDesync** – CatchDesync mode, catch desync because of sim_procedures. It might be a sign of something interesting.

**Strict = 'strict'**

**Permissive = 'permissive'**

**CatchDesync = 'catch_desync'**

**exception** angr.exploration_techniques.tracer.**TracerDesyncError**

Bases: *AngrTracerError*

An error class to report tracing Tracing desyncronization error

**class** angr.exploration_techniques.tracer.**RepHook**

Bases: object

Hook rep movs/stos to speed up constraint solving TODO: This should be made an exploration technique later

**__init__**(*mnemonic*)

**run**(*state*)

**class** angr.exploration_techniques.tracer.**Tracer**

Bases: *ExplorationTechnique*

An exploration technique that follows an angr path with a concrete input. The tracing result is the state at the last address of the trace, which can be found in the 'traced' stash.

If the given concrete input makes the program crash, you should provide crash_addr, and the crashing state will be found in the 'crashed' stash.

**Parameters**

- **trace** (list[int]) – The basic block trace.
- **resiliency** – Should we continue to step forward even if qemu and angr disagree?
- **keep_predecessors** – Number of states before the final state we should log.
- **crash_addr** – If the trace resulted in a crash, provide the crashing instruction pointer here, and the 'crashed' stash will be populated with the crashing state.
- **syscall_data** – Data related to various syscalls recorded by tracer for replaying
- **copy_states** – Whether COPY_STATES should be enabled for the tracing state. It is off by default because most tracing workloads benefit greatly from not performing copying. You want to enable it if you want to see the missed states. It will be re-added for the last 2% of the trace in order to set the predecessors list correctly. If you turn this on you may want to enable the LAZY_SOLVES option.
- **mode** – Tracing mode.
- **aslr** – Whether there are aslr slides. if not, tracer uses trace address as state address.
- **follow_unsat** – Whether unsatisfiable states should be treated as potential successors or not.

**Variables**

**predecessors** (list[*SimState*]) – A list of states in the history before the final state.

**__init__**(*trace*, *resiliency=False*, *keep_predecessors=1*, *crash_addr=None*, *syscall_data=None*, *copy_states=False*, *fast_forward_to_entry=True*, *mode='strict'*, *aslr=True*, *follow_unsat=False*)

**Parameters**

**trace** (*list[int]*)

**predecessors: list[*SimState*]**

**set_fd_data**(*fd_data*)

Set concrete bytes of various fds read by the program

**Parameters**
**fd_data** (*dict[int, bytes]*)

**classmethod crash_windup**(*state*, *crash_addr*)

### 10.17.24 angr.exploration_techniques.unique

**class** angr.exploration_techniques.unique.**UniqueSearch**

Bases: *ExplorationTechnique*

Unique Search.

Will only keep one path active at a time, any others will be deferred. The state that is explored depends on how unique it is relative to the other deferred states. A path's uniqueness is determined by its average similarity between the other (deferred) paths. Similarity is calculated based on the supplied *similarity_func*, which by default is: The (L2) distance between the counts of the state addresses in the history of the path.

**__init__**(*similarity_func=None*, *deferred_stash='deferred'*)

**Parameters**

- **similarity_func** – How to calculate similarity between two states.
- **deferred_stash** – Where to store the deferred states.

**static similarity**(*state_a*, *state_b*)

The (L2) distance between the counts of the state addresses in the history of the path. :type state_a: :param state_a: The first state to compare :type state_b: :param state_b: The second state to compare

**static sequence_matcher_similarity**(*state_a*, *state_b*)

The *difflib.SequenceMatcher* ratio between the state addresses in the history of the path. :type state_a: :param state_a: The first state to compare :type state_b: :param state_b: The second state to compare

### 10.17.25 angr.exploration_techniques.veritesting

**class** angr.exploration_techniques.veritesting.**Veritesting**

Bases: *ExplorationTechnique*

Enable veritesting. This technique, described in a paper[1] from CMU, attempts to address the problem of state explosions in loops by performing smart merging.

[1] https://users.ece.cmu.edu/~aavgerin/papers/veritesting-icse-2014.pdf

**__init__**(***options*)

## 10.18 angr.factory

**class** angr.factory.**AngrObjectFactory**

Bases: object

This factory provides access to important analysis elements.

**__init__**(*project*, *default_engine=None*)

**Parameters**
**default_engine** (*type[SimEngine] | None*)

**default_engine_factory: *type*[*SimEngine*]**

**project:** *Project*

**procedure_engine:** *ProcedureEngine*

**property default_engine**

**snippet**(*addr*, *jumpkind=None*, ***block_opts*)

**successors**(**args*, *engine=None*, ***kwargs*)

Perform execution using an engine. Generally, return a SimSuccessors object classifying the results of the run.

**Parameters**

- **state** – The state to analyze
- **engine** – The engine to use. If not provided, will use the project default.
- **addr** – optional, an address to execute at instead of the state's ip
- **jumpkind** – optional, the jumpkind of the previous exit
- **inline** – This is an inline execution. Do not bother copying the state.

Additional keyword arguments will be passed directly into each engine's process method.

**blank_state**(***kwargs*)

Returns a mostly-uninitialized state object. All parameters are optional.

**Parameters**

- **addr** – The address the state should start at instead of the entry point.
- **initial_prefix** – If this is provided, all symbolic registers will hold symbolic values with names prefixed by this string.
- **fs** – A dictionary of file names with associated preset SimFile objects.
- **concrete_fs** – bool describing whether the host filesystem should be consulted when opening files.
- **chroot** – A path to use as a fake root directory, Behaves similarly to a real chroot. Used only when concrete_fs is set to True.
- **kwargs** – Any additional keyword args will be passed to the SimState constructor.

**Returns**
The blank state.

**Return type**
*SimState*

**entry_state**(***kwargs*)

Returns a state object representing the program at its entry point. All parameters are optional.

**Parameters**

- **addr** – The address the state should start at instead of the entry point.
- **initial_prefix** – If this is provided, all symbolic registers will hold symbolic values with names prefixed by this string.
- **fs** – a dictionary of file names with associated preset SimFile objects.
- **concrete_fs** – boolean describing whether the host filesystem should be consulted when opening files.

- **chroot** – a path to use as a fake root directory, behaves similar to a real chroot. used only when concrete_fs is set to True.
- **argc** – a custom value to use for the program's argc. May be either an int or a bitvector. If not provided, defaults to the length of args.
- **args** – a list of values to use as the program's argv. May be mixed strings and bitvectors.
- **env** – a dictionary to use as the environment for the program. Both keys and values may be mixed strings and bitvectors.

**Returns**
The entry state.

**Return type**
*SimState*

**full_init_state**(***kwargs*)

Very much like *entry_state()*, except that instead of starting execution at the program entry point, execution begins at a special SimProcedure that plays the role of the dynamic loader, calling each of the initializer functions that should be called before execution reaches the entry point.

It can take any of the arguments that can be provided to `entry_state`, except for `addr`.

**call_state**(*addr*, **args*, ***kwargs*)

Returns a state object initialized to the start of a given function, as if it were called with given parameters.

**Parameters**

- **addr** – The address the state should start at instead of the entry point.
- **args** – Any additional positional arguments will be used as arguments to the function call.
- **base_state** – Use this SimState as the base for the new state instead of a blank state.
- **cc** – Optionally provide a SimCC object to use a specific calling convention.
- **ret_addr** – Use this address as the function's return target.
- **stack_base** – An optional pointer to use as the top of the stack, circa the function entry point
- **alloc_base** – An optional pointer to use as the place to put excess argument data
- **grow_like_stack** – When allocating data at alloc_base, whether to allocate at decreasing addresses
- **toc** – The address of the table of contents for ppc64
- **initial_prefix** – If this is provided, all symbolic registers will hold symbolic values with names prefixed by this string.
- **fs** – A dictionary of file names with associated preset SimFile objects.
- **concrete_fs** – bool describing whether the host filesystem should be consulted when opening files.
- **chroot** – A path to use as a fake root directory, Behaves similarly to a real chroot. Used only when concrete_fs is set to True.
- **kwargs** – Any additional keyword args will be passed to the SimState constructor.

**Returns**
The state at the beginning of the function.

**Return type**
*SimState*

The idea here is that you can provide almost any kind of python type in *args* and it'll be translated to a binary format to be placed into simulated memory. Lists (representing arrays) must be entirely elements of the same type and size, while tuples (representing structs) can be elements of any type and size. If you'd like there to be a pointer to a given value, wrap the value in a *SimCC.PointerWrapper*. Any value that can't fit in a register will be automatically put in a PointerWrapper.

If stack_base is not provided, the current stack pointer will be used, and it will be updated. If alloc_base is not provided, the current stack pointer will be used, and it will be updated. You might not like the results if you provide stack_base but not alloc_base.

grow_like_stack controls the behavior of allocating data at alloc_base. When data from args needs to be wrapped in a pointer, the pointer needs to point somewhere, so that data is dumped into memory at alloc_base. If you set alloc_base to point to somewhere other than the stack, set grow_like_stack to False so that sequential allocations happen at increasing addresses.

**simulation_manager**(*thing=None*, ***kwargs*)

Constructs a new simulation manager.

**Parameters**

- **thing** (list[*SimState*] | *SimState* | None) – What to put in the new SimulationManager's active stash (either a SimState or a list of SimStates).
- **kwargs** – Any additional keyword arguments will be passed to the SimulationManager constructor

**Returns**
The new SimulationManager

**Return type**
*SimulationManager*

Many different types can be passed to this method:

- If nothing is passed in, the SimulationManager is seeded with a state initialized for the program entry point, i.e. *entry_state()*.
- If a SimState is passed in, the SimulationManager is seeded with that state.
- If a list is passed in, the list must contain only SimStates and the whole list will be used to seed the SimulationManager.

**simgr**(**args*, ***kwargs*)

Alias for *simulation_manager* to save our poor fingers

**callable**(*addr*, *prototype=None*, *concrete_only=False*, *perform_merge=True*, *base_state=None*, *toc=None*, *cc=None*, *add_options=None*, *remove_options=None*, *techniques=None*, *step_limit=None*)

A Callable is a representation of a function in the binary that can be interacted with like a native python function.

**Parameters**

- **addr** (int | *Function*) – The address of the function to use. If you pass in the function object, we will take its addr.
- **prototype** – The prototype of the call to use, as a string or a SimTypeFunction
- **concrete_only** – Throw an exception if the execution splits into multiple states

- **perform_merge** – Merge all result states into one at the end (only relevant if concrete_only=False)
- **base_state** – The state from which to do these runs
- **toc** – The address of the table of contents for ppc64
- **cc** – The SimCC to use for a calling convention
- **step_limit** (*int | None*) – The maximum number of blocks that Callable will execute before pruning the path.
- **techniques** (*list[ExplorationTechnique] | None*)

**Returns**
A Callable object that can be used as a interface for executing guest code like a python function.

**Return type**
*angr.callable.Callable*

**cc**()

Return a SimCC (calling convention) parameterized for this project.

Relevant subclasses of SimFunctionArgument are SimRegArg and SimStackArg, and shortcuts to them can be found on this *cc* object.

For stack arguments, offsets are relative to the stack pointer on function entry.

**function_prototype**()

Return a default function prototype parameterized for this project and SimOS.

**block**(*addr*, *size=None*, *max_size=None*, *byte_string=None*, *thumb=False*, *backup_state=None*, *extra_stop_points=None*, *opt_level=None*, *num_inst=None*, *traceflags=0*, *insn_bytes=None*, *strict_block_end=None*, *collect_data_refs=False*, *cross_insn_opt=True*, *load_from_ro_regions=False*, *const_prop=False*, *initial_regs=None*, *skip_stmts=False*)

**Overloads**

- **self**, **addr** (int), **size**, **max_size**, **byte_string**, **thumb**, **backup_state**, **extra_stop_points**, **opt_level**, **num_inst**, **traceflags**, **insn_bytes**, **strict_block_end**, **collect_data_refs**, **cross_insn_opt**, **load_from_ro_regions**, **const_prop**, **initial_regs**, **skip_stmts** → Block
- **self**, **addr** (SootAddressDescriptor), **size**, **max_size**, **byte_string**, **thumb**, **backup_state**, **extra_stop_points**, **opt_level**, **num_inst**, **traceflags**, **insn_bytes**, **strict_block_end**, **collect_data_refs**, **load_from_ro_regions**, **const_prop**, **cross_insn_opt**, **skip_stmts** → SootBlock

**fresh_block**(*addr*, *size*, *backup_state=None*)

## 10.19 angr.flirt

angr.flirt.**FS**

alias of *FlirtSignature*

**class** angr.flirt.**FlirtSignature**

Bases: object

This class describes a FLIRT signature without any internal data that is only available after parsing.

__init__(*arch*, *platform*, *sig_name*, *sig_path*, *unique_strings=None*, *compiler=None*, *compiler_version=None*, *os_name=None*, *os_version=None*)

**Parameters**

- **arch** (*str*)
- **platform** (*str*)
- **sig_name** (*str*)
- **sig_path** (*str*)
- **unique_strings** (*set[str] | None*)
- **compiler** (*str | None*)
- **compiler_version** (*str | None*)
- **os_name** (*str | None*)
- **os_version** (*str | None*)

angr.flirt.**load_signature**(*sig_path*, *meta_path=None*)

Load a single FLIRT signature from a specific path.

**Parameters**

- **sig_path** (str) – Location of the FLIRT signature.
- **meta_path** (*str | None*)

**Return type**
tuple[str, *FlirtSignature*] | None

**Returns**
A FlirtSignature object if loading was successful, None otherwise.

angr.flirt.**load_signatures**(*path*, *ignore=None*)

Recursively load all FLIRT signatures under a specific path.

**Parameters**

- **path** (str) – Location of FLIRT signatures.
- **ignore** (list[str] | None) – A list of signature directory path prefixes to ignore. If None, a default list of signatures will be loaded.

**Return type**
None

### Submodules

*build_sig*

## 10.19.1 angr.flirt.build_sig

angr.flirt.build_sig.**get_basic_info**(*ar_path*)

Get basic information of the archive file.

**Return type**
dict[str, str]

**Parameters**
**ar_path** (*str*)

angr.flirt.build_sig.**get_unique_strings**(*ar_path*)

For Linux libraries, this method requires ar (from binutils), nm (from binutils), and strings.

**Return type**
list[str]

**Parameters**
**ar_path** (*str*)

angr.flirt.build_sig.**run_pelf**(*pelf_path*, *ar_path*, *output_path*)

**Parameters**

- **pelf_path** (*str*)
- **ar_path** (*str*)
- **output_path** (*str*)

angr.flirt.build_sig.**run_sigmake**(*sigmake_path*, *sig_name*, *pat_path*, *sig_path*)

**Parameters**

- **sigmake_path** (*str*)
- **sig_name** (*str*)
- **pat_path** (*str*)
- **sig_path** (*str*)

angr.flirt.build_sig.**process_exc_file**(*exc_path*)

We are doing the stupidest thing possible: For each batch of conflicts, we pick the most likely result based on a set of predefined rules.

TODO: Add caller-callee-based de-duplication.

**Parameters**
**exc_path** (*str*)

angr.flirt.build_sig.**main**()

## 10.20 angr.keyed_region

**class** angr.keyed_region.**StoredObject**

Bases: object

**__init__**(*start*, *obj*, *size*)

**start**

**obj**

**size:** *UnknownSize* | int

**property obj_id**

class angr.keyed_region.**RegionObject**

Bases: object

Represents one or more objects occupying one or more bytes in KeyedRegion.

**__init__**(*start*, *size*, *objects=None*)

**start**

**size**

**stored_objects**

**property is_empty**

**property end**

**property internal_objects**

**includes**(*offset*)

**split**(*split_at*)

**add_object**(*obj*)

**set_object**(*obj*)

**remove_object**(*obj*)

**Return type**
None

**copy**()

class angr.keyed_region.**KeyedRegion**

Bases: object

KeyedRegion keeps a mapping between stack offsets and all objects covering that offset. It assumes no variable in this region overlap with another variable in this region.

Registers and function frames can all be viewed as a keyed region.

**__init__**(*tree=None*, *phi_node_contains=None*, *canonical_size=8*)

**copy**()

**merge**(*other*, *replacements=None*)

Merge another KeyedRegion into this KeyedRegion.

**Parameters**
**other** (KeyedRegion) – The other instance to merge with.

**Returns**
None

**merge_to_top**(*other*, *replacements=None*, *top=None*)

Merge another KeyedRegion into this KeyedRegion, but mark all variables with different values as TOP.

**Parameters**

- **other** – The other instance to merge with.
- **replacements**

**Returns**
self

`replace`(*replacements*)

Replace variables with other variables.

**Parameters**
**replacements** (*dict*) – A dict of variable replacements.

**Returns**
self

`dbg_repr`()

Get a debugging representation of this keyed region. :return: A string of debugging output.

`add_variable`(*start*, *variable*)

Add a variable to this region at the given offset.

**Parameters**

- **start** (*int*)
- **variable** (SimVariable)

**Returns**
None

`add_object`(*start*, *obj*, *object_size*)

Add/Store an object to this region at the given offset.

**Parameters**

- **start**
- **obj**
- **object_size** (*int*) – Size of the object

**Returns**

`set_variable`(*start*, *variable*)

Add a variable to this region at the given offset, and remove all other variables that are fully covered by this variable.

**Parameters**

- **start** (*int*)
- **variable** (SimVariable)

**Returns**
None

`set_object`(*start*, *obj*, *object_size*)

Add an object to this region at the given offset, and remove all other objects that are fully covered by this object.

**Parameters**

- **start**
- **obj**
- **object_size**

**Returns**

**remove_variable**(*start*, *variable*)

Remove a variable that was added at the given offset. Other variables are untouched.

**Return type**
None

**remove_object**(*start*, *obj*, *object_size*)

**Return type**
None

**get_base_addr**(*addr*)

Get the base offset (the key we are using to index objects covering the given offset) of a specific offset.

**Parameters**
**addr** (*int*)

**Returns**

**Return type**
int or None

**get_variables_by_offset**(*start*)

Find variables covering the given region offset.

**Parameters**
**start** (*int*)

**Returns**
A set of variables.

**Return type**
set

**get_objects_by_offset**(*start*)

Find objects covering the given region offset.

**Parameters**
**start**

**Returns**

**get_all_variables**()

Get all variables covering the current region.

**Returns**
A set of all variables.

## 10.21 angr.knowledge_base

Representing the artifacts of a project.

**class** angr.knowledge_base.**KnowledgeBase**

Bases: object

Represents a "model" of knowledge about an artifact.

Contains things like a CFG, data references, etc.

**functions:** *FunctionManager*

**comments:** *Comments*

**bookmarks:** *Bookmarks*

**variables:** *VariableManager*

**defs:** *KeyDefinitionManager*

**cfgs:** *CFGManager*

**types:** *TypesStore*

**propagations:** *PropagationManager*

**xrefs:** *XRefManager*

**decompilations:** *StructuredCodeManager*

**obfuscations:** *Obfuscations*

**rtdb:** *RuntimeDb*

**__init__**(*project*, *obj=None*, *name=None*)

**property callgraph**

**property unresolved_indirect_jumps**

**property resolved_indirect_jumps**

**has_plugin**(*name*)

**get_plugin**(*name*)

**register_plugin**(*name*, *plugin*)

**release_plugin**(*name*)

**get_knowledge**(*requested_plugin_cls*)

Type inference safe method to request a knowledge base plugin Explicitly passing the type of the requested plugin achieves two things: 1. Every location using this plugin can be easily found with an IDE by searching explicit references to the type 2. Basic type inference can deduce the result type and properly type check usages of it

If there isn't already an instance of this class None will be returned to make it clear to the caller that there is no existing knowledge of this type yet. The code that initially creates this knowledge should use the *register_plugin* method to register the initial knowledge state :type requested_plugin_cls: `type`[`TypeVar`(K, bound= *KnowledgeBasePlugin*)] :param requested_plugin_cls:

**Return type**
`Optional`[`TypeVar`(K, bound= *KnowledgeBasePlugin*)]

**Returns**
Instance of the requested plugin class or null if it is not a known plugin

**Parameters**
**requested_plugin_cls** (`type`*[K]*)

**request_knowledge**(*requested_plugin_cls*)

**Return type**
TypeVar(K, bound= *KnowledgeBasePlugin*)

**Parameters**
**requested_plugin_cls** (type*[K]*)

## 10.22 angr.knowledge_plugins

**class** angr.knowledge_plugins.**Bookmark**

Bases: object

A user-marked address with an optional label.

**__init__**(*addr*, *label=''*, *created_at=None*)

**Parameters**

- **addr** (*int*)
- **label** (*str*)
- **created_at** (*float | None*)

**Return type**
None

**addr**

**label**

**created_at**

**class** angr.knowledge_plugins.**Bookmarks**

Bases: *KnowledgeBasePlugin*, list

Tracks bookmarks as a list of Bookmark, in creation order.

**get_bookmark**(*addr*)

**Return type**
*Bookmark* | None

**Parameters**
**addr** (*int*)

**has_bookmark**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

**add_bookmark**(*addr*, *label=''*)

Bookmark addr, or relabel the bookmark already there if label is non-empty. Returns the bookmark and whether it was newly created.

**Return type**
tuple[*Bookmark*, bool]

**Parameters**

- **addr** (*int*)
- **label** (*str*)

**remove_bookmark**(*bookmark*)

Remove bookmark if present. Returns whether it was removed.

**Return type**
bool

**Parameters**
**bookmark** (Bookmark)

**toggle_bookmark**(*addr*, *label=''*)

Add a bookmark at addr, or remove the one already there. Returns the new one, if any.

**Return type**
*Bookmark* | None

**Parameters**

- **addr** (*int*)
- **label** (*str*)

**sorted_bookmarks**()

**Return type**
list[*Bookmark*]

**next_bookmark**(*after_addr*)

The next bookmark by address, wrapping around.

**Return type**
*Bookmark* | None

**Parameters**
**after_addr** (*int | None*)

**class** angr.knowledge_plugins.**CFGManager**

Bases: *KnowledgeBasePlugin*

This is the CFG manager, it manages CFGs

**__init__**(*kb*)

**new_model**(*prefix*, *addr_type='int'*)

**Parameters**
**addr_type** (*Literal['int', 'block_id', 'soot']*)

**copy**()

**get_most_accurate**()

**Return type**
*CFGModel* | None

**Returns**
The most accurate CFG present in the CFGManager, or None if it does not hold any.

class angr.knowledge_plugins.**CallsitePrototypes**

Bases: *KnowledgeBasePlugin*

CallsitePrototypes manages callee prototypes at call sites.

**__init__**(*kb*)

**set_prototype**(*callsite_block_addr*, *cc*, *prototype*, ***, *kind=CallsitePrototypeKind.INFERRED*, *manual=False*, *propagated=False*)

**Return type**
None

**Parameters**

- **callsite_block_addr** (*int*)
- **cc** (SimCC)
- **prototype** (SimTypeFunction)
- **kind** (CallsitePrototypeKind)
- **manual** (*bool*)
- **propagated** (*bool*)

**get_cc**(*callsite_block_addr*, ***, *kind=CallsitePrototypeKind.INFERRED*)

**Return type**
*SimCC* | None

**Parameters**

- **callsite_block_addr** (*int*)
- **kind** (CallsitePrototypeKind)

**get_prototype**(*callsite_block_addr*, ***, *kind=CallsitePrototypeKind.INFERRED*)

**Return type**
*SimTypeFunction* | None

**Parameters**

- **callsite_block_addr** (*int*)
- **kind** (CallsitePrototypeKind)

**is_prototype_manual**(*callsite_block_addr*)

**Return type**
bool | None

**Parameters**
**callsite_block_addr** (*int*)

**is_prototype_certain**(*callsite_block_addr*)

**Return type**
bool | None

**Parameters**
**callsite_block_addr** (*int*)

**has_some_prototype**(*callsite_block_addr*)

**Return type**
bool

**Parameters**
**callsite_block_addr** (*int*)

**has_prototype**(*callsite_block_addr*, ***, *kind=CallsitePrototypeKind.INFERRED*)

**Return type**
bool

**Parameters**

- **callsite_block_addr** (*int*)
- **kind** (CallsitePrototypeKind)

**copy**()

**class** angr.knowledge_plugins.**Comment**

Bases: object

A comment together with its display metadata. Rebuilt on demand from the knowledge base.

**__init__**(*addr*, *text*, *kind*, *func_addr=None*, *func_name=None*)

**Parameters**

- **addr** (*int*)
- **text** (*str*)
- **kind** (CommentKind)
- **func_addr** (*int | None*)
- **func_name** (*str | None*)

**Return type**
None

**addr**

**text**

**kind**

**func_addr**

**func_name**

**property first_line: str**

**class** angr.knowledge_plugins.**CommentKind**

Bases: IntEnum

How a comment is displayed.

PLAIN shows only at its own address. FUNCTION renders as a header block above the function, which is also what the decompiler does with a comment on a function entry. REPEATABLE additionally shows at every location referencing the commented address.

PLAIN = 0

FUNCTION = 1

REPEATABLE = 2

property display_name: str

__new__(*value*)

class angr.knowledge_plugins.Comments

Bases: *KnowledgeBasePlugin*, dict

Tracks comments via a Dict of Address -> Text, along with a per-address CommentKind.

property kinds: dict[int, *CommentKind*]

Explicitly-set comment kinds. Addresses not present here use kind_of() defaults.

kind_of(*addr*)

**Return type**
*CommentKind*

**Parameters**
**addr** (*int*)

set_kind(*addr*, *kind*)

Set the display kind for the comment at addr, or reset it to the default with None.

**Return type**
None

**Parameters**

- **addr** (*int*)
- **kind** (CommentKind | *None*)

is_function_entry(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

function_comment(*func_addr*)

The comment rendered as a function's header block, if any.

**Return type**
str | None

**Parameters**
**func_addr** (*int*)

inline_comment(*addr*)

The comment rendered next to the instruction at addr. A comment that already renders as a function header block is not repeated inline.

**Return type**
str | None

**Parameters**
**addr** (*int*)

`iter_comments()`

All comments in address order, with their kinds and owning functions.

**Return type**
Iterator[*Comment*]

`repeatable_comments_at(ins_addr)`

Repeatable comments that should show at `ins_addr` because it references their address.

**Return type**
list[tuple[int, str]]

**Parameters**
**ins_addr** (*int*)

`invalidate()`

Drop the cached repeatable-comment map, e.g. after xrefs or functions change.

**Return type**
None

`class angr.knowledge_plugins.CustomStrings`

Bases: *KnowledgeBasePlugin*

Store new strings that are recovered during various analysis. Each string has a unique ID associated.

`__init__(kb)`

`allocate(s)`

**Return type**
int

**Parameters**
**s** (bytes)

`copy()`

`class angr.knowledge_plugins.Data`

Bases: *KnowledgeBasePlugin*

The knowledge what purpose this plugin serves has been lost to the passing of time but the linter does not care for these failures of mere mortals and demands a docstring anyway. The pact has been made, and no violations of the rules will be tolerated, even if the spirit does not match the letter. Making the plugin smaller has only increased the weight of the failure, and thus this file has drawn its ire.

The only thing left to do is to attempt to find meaning in the meaninglessness, as the only act of rebellion against the uncaring forces that bind us. For is this not what being human is all about?

`copy()`

`class angr.knowledge_plugins.DebugVariableManager`

Bases: *KnowledgeBasePlugin*

Structure to manage and access variables with different visibility scopes.

__init__(*kb*)

**Parameters**
**kb** (KnowledgeBase)

from_name_and_pc(*var_name*, *pc_addr*)

Get a variable from its string in the scope of pc.

**Return type**
Variable

**Parameters**

- **var_name** (*str*)
- **pc_addr** (*int*)

from_name(*var_name*)

Get the variable container for all variables named var_name

**Parameters**
**var_name** (str) – name for a variable

**Return type**
*DebugVariableContainer*

add_variable(*cle_var*, *low_pc*, *high_pc*)

Add/load a variable

**Parameters**

- **cle_variable** – The variable to add
- **low_pc** (int) – Start of the visibility scope of the variable as program counter address (rebased)
- **high_pc** (int) – End of the visibility scope of the variable as program counter address (rebased)
- **cle_var** (*Variable*)

add_variable_list(*vlist*, *low_pc*, *high_pc*)

Add all variables in a list with the same visibility range

**Parameters**

- **vlist** (list[Variable]) – A list of cle variables to add
- **low_pc** (int) – Start of the visibility scope as program counter address (rebased)
- **high_pc** (int) – End of the visibility scope as program counter address (rebased)

load_from_dwarf(*elf_object=None*, *cu=None*)

Automatically load all variables (global/local) from the DWARF debugging info

**Parameters**

- **elf_object** (ELF) – Optional, when only one elf object should be considered (e.g. p.loader.main_object)
- **cu** (CompilationUnit) – Optional, when only one compilation unit should be considered

class angr.knowledge_plugins.**Function**

Bases: *Serializable*

A representation of a function and various information about it.

**Variables**

- **meta_only** (bool) – Whether this function only contains meta-information and in read-only mode.
- **_dirty** (bool) – Whether this function has been modified since last serialization.
- **evicted** (bool) – Whether this function has been evicted from FunctionManager to external storage.

**__init__**(*function_manager*, *addr*, *name=None*, *syscall=None*, *is_simprocedure=None*, *binary_name=None*, *is_plt=None*, *returning=None*, *alignment=False*, *calling_convention=None*, *prototype=None*, *prototype_libname=None*, *prototype_source=None*, *is_prototype_guessed=True*)

Function constructor. If the optional parameters are not provided, they will be automatically determined upon the creation of a Function object.

**Parameters**

- **addr** (int) – The address of the function.
- **name** (*str*) – The name of the function.
- **syscall** (*bool*) – Whether this function is a syscall or not.
- **is_simprocedure** (bool | None) – Whether this function is a SimProcedure or not.
- **binary_name** (*str*) – Name of the binary where this function is.
- **is_plt** (bool | None) – If this function is a PLT entry.
- **returning** (*bool*) – If this function returns.
- **alignment** (*bool*) – If this function acts as an alignment filler. Such functions usually only contain nops.
- **function_manager** (FunctionManager | *None*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **prototype_libname** (*str* | *None*)
- **prototype_source** (PrototypeSource | *None*)
- **is_prototype_guessed** (*bool*)

**transition_graph**

**normalized**

**addr**

**startpoint**

**bp_on_stack**

**retaddr_on_stack**

**sp_delta**

**tags**

**ran_cca**

**meta_only:** *bool*

**evicted:** *bool*

**is_default_name**

**previous_names**

**binary_name**

**property name**

**property from_signature:** *str | None*

**property project:** *Project | None*

**property returning**

**property calling_convention:** *SimCC | None*

**property prototype:** *SimTypeFunction | None*

**property prototype_libname**

**property is_prototype_guessed:** *bool*

**property is_prototype_groundtruth:** *bool*

True if the prototype comes from outside of the decompiler (SimProcedures, signatures, or the user) and may be fed back into type inference as ground truth. Prototypes inferred by the decompiler itself are excluded so that re-decompiling a function does not freeze its own earlier guess.

**property prototype_source:** *PrototypeSource*

**property info:** *FunctionInfo*

**property is_plt:** *bool*

**property is_simprocedure:** *bool*

**property is_syscall:** *bool*

**property is_alignment:** *bool*

**property blocks**

An iterator of all local blocks in the current function.

**Returns**
angr.lifter.Block instances.

**property code_nodes:** *dict[int, CodeNode]*

**property cyclomatic_complexity**

The cyclomatic complexity of the function.

Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. It is computed using the formula: M = E - N + 2P, where E = the number of edges in the graph, N = the number of nodes in the graph, P = the number of connected components.

The cyclomatic complexity value is lazily computed and cached for future use. Initially this value is None until it is computed for the first time

**Returns**
The cyclomatic complexity of the function.

**Return type**
int

**property xrefs: Iterator[*XRef*]**

An iterator of all xrefs of the current function.

**Returns**
angr.knowledge_plugins.xrefs.xref.XRef instances.

**property block_addrs**

An iterator of all local block addresses in the current function.

**Returns**
block addresses.

**property block_addrs_set**

Return a set of block addresses for a better performance of inclusion tests.

**Returns**
A set of block addresses.

**Return type**
set

**get_block**(*addr*, *size=None*, *byte_string=None*)

Getting a block out of the current function.

**Parameters**

- **addr** (int) – The address of the block.
- **size** (int | None) – The size of the block. This is optional. If not provided, angr will load
- **byte_string** (bytes | None)

**Returns**

**get_block_size**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*int*)

**property nodes: Iterable[*CodeNode*]**

`get_node`(*addr*)

**Return type**
*BlockNode* | None

`property has_unresolved_jumps`

`property has_unresolved_calls`

`property operations`
All of the operations that are done by this functions.

`property code_constants`
All of the constants that are used by this functions's code.

`classmethod parse_from_cmessage`(*cmsg*, ***kwargs*)

**Parameters**
**cmsg**

**Return Function**
The function instantiated out of the cmsg data.

`string_references`(*minimum_length=2*)
All of the constant string references used by this function.

**Parameters**
**minimum_length** – The minimum length of strings to find (default is 1)

**Returns**
A generator yielding tuples of (address, string) where is address is the location of the string in memory.

`property local_runtime_values`
Tries to find all runtime values of this function which do not come from inputs. These values are generated by starting from a blank state and reanalyzing the basic blocks once each. Function calls are skipped, and back edges are never taken so these values are often unreliable, This function is good at finding simple constant addresses which the function will use or calculate.

**Returns**
a set of constants

`property num_arguments`

`property endpoints`

`property endpoints_with_type`

`property ret_sites`

`property jumpout_sites`

`property retout_sites`

`property callout_sites`

`property size`

`property binary`
Get the object this function belongs to. :return: The object this function belongs to.

**property offset:** int

the function's binary offset (i.e., non-rebased address)

**Type**
return

**property symbol:** Symbol | None

the function's Symbol, if any

**Type**
return

**property pseudocode:** str | None

the function's pseudocode

**Type**
return

**property dirty:** bool

**mark_dirty**()

**Return type**
None

**add_jumpout_site**(*node*)

Add a custom jumpout site.

**Parameters**
**node** (*CodeNode*) – The address of the basic block that control flow leaves during this transition.

**Returns**
None

**add_retout_site**(*node*)

Add a custom retout site.

Retout (returning to outside of the function) sites are very rare. It mostly occurs during CFG recovery when we incorrectly identify the beginning of a function in the first iteration, and then correctly identify that function later in the same iteration (function alignments can lead to this bizarre case). We will mark all edges going out of the header of that function as a outside edge, because all successors now belong to the incorrectly-identified function. This identification error will be fixed in the second iteration of CFG recovery. However, we still want to keep track of jumpouts/retouts during the first iteration so other logic in CFG recovery still work.

**Parameters**
**node** (*CodeNode*) – The address of the basic block that control flow leaves the current function after a call.

**Returns**
None

**update_func_block_count**()

Update the cached block count of this function in the function manager.

**Return type**
None

`mark_nonreturning_calls_endpoints()`

Iterate through all call edges in transition graph. For each call a non-returning function, mark the source basic block as an endpoint.

This method should only be executed once all functions are recovered and analyzed by CFG recovery, so we know whether each function returns or not.

**Returns**
None

`get_call_sites()`

Gets a list of all the basic blocks that end in calls.

**Return type**
`Iterable[int]`

**Returns**
A view of the addresses of the blocks that end in calls.

`get_call_target(callsite_addr)`

Get the target of a call.

**Parameters**
**callsite_addr** – The address of a basic block that ends in a call.

**Returns**
The target of said call, or None if callsite_addr is not a callsite.

`get_call_return(callsite_addr)`

Get the hypothetical return address of a call.

**Parameters**
**callsite_addr** – The address of the basic block that ends in a call.

**Returns**
The likely return target of said call, or None if callsite_addr is not a callsite.

`property graph: networkx.DiGraph[CodeNode]`

Get a local transition graph. A local transition graph is a transition graph that only contains nodes that belong to the current function. All edges, except for the edges going out from the current function or coming from outside the current function, are included.

The generated graph is cached in self._local_transition_graph.

**Returns**
A local transition graph.

**Return type**
networkx.DiGraph

`graph_ex(exception_edges=True)`

Get a local transition graph with a custom configuration. A local transition graph is a transition graph that only contains nodes that belong to the current function. This method allows user to exclude certain types of edges together with the nodes that are only reachable through such edges, such as exception edges.

The generated graph is not cached.

**Parameters**
**exception_edges** (*bool*) – Should exception edges and the nodes that are only reachable through exception edges be kept.

**Returns**
A local transition graph with a special configuration.

**Return type**
networkx.DiGraph

`transition_graph_ex`(*exception_edges=True*)

Get a transition graph with a custom configuration. This method allows user to exclude certain types of edges together with the nodes that are only reachable through such edges, such as exception edges.

The generated graph is not cached.

**Parameters**
**exception_edges** (*bool*) – Should exception edges and the nodes that are only reachable through exception edges be kept.

**Returns**
A local transition graph with a special configuration.

**Return type**
networkx.DiGraph

`subgraph`(*ins_addrs*)

Generate a sub control flow graph of instruction addresses based on self.graph

**Parameters**
**ins_addrs** (*iterable*) – A collection of instruction addresses that should be included in the subgraph.

**Return networkx.DiGraph**
A subgraph.

`instruction_size`(*insn_addr*)

Get the size of the instruction specified by *insn_addr*.

**Parameters**
**insn_addr** (*int*) – Address of the instruction

**Return int**
Size of the instruction in bytes, or None if the instruction is not found.

`addr_to_instruction_addr`(*addr*)

Obtain the address of the instruction that covers @addr.

**Parameters**
**addr** (*int*) – An address.

**Returns**
Address of the instruction that covers @addr, or None if this addr is not covered by any instruction of this function.

**Return type**
int or None

`dbg_print`()

Returns a representation of the list of basic blocks in this function.

`dbg_draw`(*filename*)

Draw the graph and save it to a PNG file.

`property arguments`

**property has_return**

**property callable**

**normalize**()

Make sure all basic blocks in the transition graph of this function do not overlap. You will end up with a CFG that IDA Pro generates.

This method does not touch the CFG result. You may call CFG{Emulated, Fast}.normalize() for that matter.

**Returns**
None

**find_declaration**(*ignore_binary_name=False*, *binary_name_hint=None*)

Find the most likely function declaration from the embedded collection of prototypes, set it to self.prototype, and update self.calling_convention with the declaration.

**Parameters**

- **ignore_binary_name** (bool) – Do not rely on the executable or library where the function belongs to determine its source library. This is useful when working on statically linked binaries (because all functions will belong to the main executable). We will search for all libraries in angr to find the first declaration match.
- **binary_name_hint** (str | None) – Substring of the library name where this function might be originally coming from. Useful for FLIRT-identified functions in statically linked binaries.

**Return type**
bool

**Returns**
True if a declaration is found and self.prototype and self.calling_convention are updated. False if we fail to find a matching function declaration, in which case self.prototype or self.calling_convention will be kept untouched.

**is_rust_function**()

Determines if the function name follows Rust mangling conventions.

**property demangled_name**

**property short_name**

**get_unambiguous_name**(*display_name=None*)

Get a disambiguated function name.

**Parameters**
**display_name** (str | None) – Name to display, otherwise the function name.

**Return type**
str

**Returns**

The function name in one of the following forms:

- ::<name> when the function binary is the main object.
- ::<obj>::<name> when the function binary is not the main object.
- ::<addr>::<name> when the function binary is an unnamed non-main object, or when multiple functions with the same name are defined in the function binary.

**apply_definition**(*definition*, *calling_convention=None*)

**Return type**
None

**Parameters**

- **definition** (*str*)
- **calling_convention** (SimCC | type[SimCC] | *None*)

**functions_reachable**()

**Return type**
set[*Function*]

**Returns**
The set of all functions that can be reached from the function represented by self.

**holes**(*min_size=8*)

Find the number of non-consecutive areas in the function that are at least *min_size* bytes large.

**Return type**
int

**Parameters**
**min_size** (*int*)

**copy**()

**pp**(***kwargs*)

Pretty-print the function disassembly.

**class** angr.knowledge_plugins.**FunctionManager**

Bases: *KnowledgeBasePlugin*, Mapping[K, *Function*], Generic

When cache_limit is set, the FunctionManager uses a SpillingFunctionDict that implements an LRU cache keeping only the most recently accessed N functions in memory, spilling others to an LMDB database on disk. This allows working with binaries that have more functions than can fit in memory.

**Parameters**
**cache_limit** (int | None) – Maximum number of functions to keep in memory. None means unlimited (no eviction). Default is None.

**__init__**(*kb*, *cache_limit=None*)

**Parameters**

- **kb** (KnowledgeBase)
- **cache_limit** (*int* | *None*)

**copy**()

**clear**()

**get_default_cache_limit**(*max_limit=5000*)

Get the default function cache limit based on the size of the binary.

**Return type**
int | None

**Returns**
The default cache limit; None means unlimited.

**Parameters**
**max_limit** (*int*)

**is_plt_cached**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

**get_binary_name_cached**(*addr*)

**Return type**
str | None

**Parameters**
**addr** (*int*)

**function_name_changed**(*addr*, *old_name*, *new_name*)

Notify the FunctionManager that a function's name has changed.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **old_name** (str | None) – Old name of the function, or None if there is no old name.
- **new_name** (str) – New name of the function.

**Return type**
None

**get_by_addr**(*addr*, *meta_only=False*)

**Return type**
*Function*

**Parameters**
**meta_only** (*bool*)

**get_by_name**(*name*, *check_previous_names=False*)

**Return type**
Generator[*Function*]

**Parameters**

- **name** (*str*)
- **check_previous_names** (*bool*)

**get_addrs_by_name**(*name*, *check_previous_names=False*)

**Return type**
set[int]

**Parameters**

- **name** (*str*)
- **check_previous_names** (*bool*)

**contains_addr**(*addr*)

Decide if an address is handled by the function manager.

Note: this function is non-conformant with python programming idioms, but its needed for performance reasons.

**Parameters**
**addr** (*int*) – Address of the function.

**ceiling_addr**(*addr*)

Return the function who has the least address that is greater than or equal to *addr*.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – The address to query.

**Return type**
Optional[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
A Function instance, or None if there is no other function after *addr*.

**ceiling_func**(*addr*)

Return the function who has the least address that is greater than or equal to *addr*.

**Parameters**
**addr** (*int*) – The address to query.

**Return type**
*Function* | None

**Returns**
A Function instance, or None if there is no other function after *addr*.

**floor_addr**(*addr*)

Return the function who has the greatest address that is less than or equal to *addr*.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – The address to query.

**Return type**
Optional[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
An address, or None if there is no other function before *addr*.

**floor_func**(*addr*)

Return the function who has the greatest address that is less than or equal to *addr*.

**Parameters**
**addr** (*int*) – The address to query.

**Returns**
A Function instance, or None if there is no other function before *addr*.

**Return type**
*Function* or None

**query**(*query*, *check_previous_names=False*)

Query for a function using selectors to disambiguate. Supported variations:

::<name> Function <name> in the main object ::<addr>::<name> Function <name> at <addr> ::<obj>::<name> Function <name> in <obj>

**Return type**
*Function* | None

**Parameters**

- **query** (*str*)
- **check_previous_names** (*bool*)

**function**(*addr=None*, *name=None*, *check_previous_names=False*, *create=False*, *syscall=False*, *plt=None*)

Get a function object from the function manager.

Pass either *addr* or *name* with the appropriate values.

**Parameters**

- **addr** (Optional[TypeVar(K, int, SootMethodDescriptor)]) – Address of the function.
- **name** (str | None) – Name of the function.
- **create** (bool) – Whether to create the function or not if the function does not exist.
- **syscall** (bool) – True to create the function as a syscall, False otherwise.
- **plt** (bool | None) – True to find the PLT stub, False to find a non-PLT stub, None to disable this restriction.
- **check_previous_names** (*bool*)

**Returns**
The Function instance, or None if the function is not found and create is False.

**Return type**
*Function* | None

**dbg_draw**(*prefix='dbg_function_'*)

**rebuild_callgraph**()

**set_function_returning**(*addr*, *v*)

**Return type**
None

**Parameters**

- **addr** (*K*)
- **v** (*bool | None*)

**nonreturning_func_addrs**()

Yield all non-returning function addresses.

**Return type**
Generator[int]

**unknown_returning_func_addrs**()

Yield all function addresses with unknown returning status.

**Return type**
Generator[int]

**is_func_nonreturning**(*addr*)

Check if a function is non-returning.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
bool

**Returns**
True if non-returning, False if returning or unknown.

**is_func_returning_unknown**(*addr*)

Check if a function's returning status is unknown.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
bool

**Returns**
True if returning status is unknown, False otherwise.

**get_func_block_count**(*addr*)

Get the number of blocks in a function.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
int | None

**Returns**
Number of blocks, or None if unknown.

**set_func_block_count**(*addr*, *count*)

Set the number of blocks in a function.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **count** (int) – Number of blocks.

**Return type**
None

**Returns**
None

**get_func_addr_with_block_count**(*block_count*)

Get the addresses of all functions that have exactly `block_count` blocks.

**Parameters**
**block_count** (int) – The number of blocks to match.

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
The set of matching function addresses.

`get_func_name`(*addr*)

Get the name of a function without loading the (possibly spilled) Function object.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
str | None

**Returns**
The function's name, or None if unknown.

`get_from_signature`(*addr*)

Get the `from_signature` value of a function without loading the (possibly spilled) Function object.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
str | None

**Returns**
The function's `from_signature` value (e.g. `"flirt"`), or None.

`set_from_signature`(*addr*, *from_signature*)

Update the cached `from_signature` value for a function. Called by Function.from_signature's setter.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **from_signature** (str | None) – The new `from_signature` value.

**Return type**
None

**Returns**
None

`get_all_funcaddrs_from_signature`(*from_signature='flirt'*)

Get the addresses of all functions whose `from_signature` matches, without loading any Function object.

**Parameters**
**from_signature** (str) – The `from_signature` value to match (default `"flirt"`).

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
The set of matching function addresses.

`get_key_func_addrs`(*func_type*)

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Parameters**
**func_type** (str)

`get_key_func_type_and_addrs`()

Get all cached key functions grouped by key-function type (e.g. `"alloca_probe"`, `"jmp_rax"`).

**Return type**
dict[str, set[TypeVar(K, int, SootMethodDescriptor)]]

**Returns**
A mapping from key-function type to the set of function addresses of that type.

**add_key_func_addr**(*func_type*, *addr*)

**Return type**
None

**Parameters**

- **func_type** (*str*)
- **addr** (*K*)

**index_key_func_addrs**(*func*)

Record every key-function flag that `func.info` carries in the key-function address cache.

**Return type**
None

**Parameters**
**func** (Function)

**property cache_limit: int | None**

Get the maximum number of functions to keep in memory. None means unlimited (no eviction).

**property cached_function_count: int**

Return the number of functions currently in memory.

**property spilled_function_count: int**

Return the number of functions currently spilled to LMDB.

**property total_function_count: int**

Return the total number of functions (in memory + spilled).

**class** angr.knowledge_plugins.**IndirectJumps**

Bases: *KnowledgeBasePlugin*, dict

This plugin tracks the targets of indirect jumps

**update_resolved_addrs**(*indirect_address*, *resolved_addresses*)

**Parameters**

- **indirect_address** (*int*)
- **resolved_addresses** (*list[int]*)

**class** angr.knowledge_plugins.**KeyDefinitionManager**

Bases: *KnowledgeBasePlugin*

KeyDefinitionManager manages and caches reaching definition models for each function.

For each function, by default we cache the entire reaching definitions model with observed results at the following locations: - Before each call instruction: ('insn', address of the call instruction, OP_BEFORE) - After returning from each call: ('node', address of the block that ends with a call, OP_AFTER)

**__init__**(*kb*)

**Parameters**
**kb** (KnowledgeBase)

**has_model**(*func_addr*)

**Parameters**
**func_addr** (*int*)

**get_model**(*func_addr*)

**Parameters**
**func_addr** (*int*)

**copy**()

**Return type**
*KeyDefinitionManager*

**class** angr.knowledge_plugins.**KnowledgeBasePlugin**

Bases: object

**__init__**(*kb*)

**Parameters**
**kb** (KnowledgeBase)

**copy**()

**set_kb**(*kb*)

Invoked after __setstate__() is called.

**Parameters**
**kb** (KnowledgeBase)

**static register_default**(*name*, *cls*)

**class** angr.knowledge_plugins.**Labels**

Bases: *KnowledgeBasePlugin*

**__init__**(*kb*)

**items**()

**get**(*addr*, *default=None*)

Get a label as string for a given address Same as .labels[x]

**lookup**(*name*, *default=None*)

Returns an address to a given label To show all available labels, iterate over .labels or list(b.kb.labels)

**copy**()

**get_unique_label**(*label*)

Get a unique label name from the given label name.

**Parameters**
**label** (*str*) – The desired label name.

**Returns**
A unique label name.

**class** angr.knowledge_plugins.**Obfuscations**

Bases: *KnowledgeBasePlugin*

Store discovered information and artifacts about (string) obfuscation techniques in the project.

__init__(*kb*)

copy()

class angr.knowledge_plugins.PatchManager

Bases: *KnowledgeBasePlugin*

A placeholder-style implementation for a binary patch manager. This class should be significantly changed in the future when all data about loaded binary objects are loaded into angr knowledge base from CLE. As of now, it only stores byte-level replacements.

Patches should not overlap, but it's user's responsibility to check for and avoid overlapping patches.

__init__(*kb*)

add_patch(*addr*, *new_bytes*, *comment=None*)

**Parameters**
**comment** (*str | None*)

add_patch_obj(*patch*)

**Parameters**
**patch** (Patch)

remove_patch(*addr*)

patch_addrs()

get_patch(*addr*)

Get patch at the given address.

**Parameters**
**addr** (*int*) – The address of the patch.

**Returns**
The patch if there is one starting at the address, or None if there isn't any.

**Return type**
*Patch* or None

get_all_patches(*addr*, *size*)

Retrieve all patches that cover a region specified by [addr, addr+size).

**Parameters**

- **addr** (*int*) – The address of the beginning of the region.
- **size** (*int*) – Size of the region.

**Returns**
A list of patches.

**Return type**
list

keys()

items()

values()

copy()

static overlap(*a0*, *a1*, *b0*, *b1*)

apply_patches_to_binary(*binary_bytes=None*, *patches=None*)

**Return type**
bytes

**Parameters**

- **binary_bytes** (bytes | *None*)
- **patches** (*list[Patch]* | *None*)

apply_patches_to_state(*state*)

property patched_entry_state

class angr.knowledge_plugins.**PropagationManager**

Bases: *KnowledgeBasePlugin*

Manages the results of Propagator, including intermediate results for unfinished Propagation runs.

__init__(*kb*)

exists(*prop_key*)

Internal function to check if a func, specified as a CodeLocation exists in our known propagations

**Parameters**
**prop_key** (tuple) – A key of the propagation result.

**Return type**
bool

**Returns**
Whether such a key exists or not.

update(*prop_key*, *model*)

Add the replacements to known propagations

**Parameters**

- **prop_key** (tuple) – A key of the propagation result.
- **model** (*PropagationModel*) – The propagation result to store

**Return type**
None

get(*prop_key*, *default=None*)

Gets the replacements for a specified function location. If the replacement does not exist in the known propagations, it returns None.

**Parameters**

- **prop_key** – A key of the propagation result.
- **default** – The default value to return if the prop_key does not exist in the cache.

**Return type**
*PropagationModel*

**Returns**
Dict or None

copy()

discard_by_prefix(*prefix*)

**Parameters**
**prefix** (*str*)

class angr.knowledge_plugins.**RuntimeDb**

Bases: *KnowledgeBasePlugin*

External storage-backed database for angr knowledge base plugins.

__init__(*kb*, *lmdb_path=None*)

**Parameters**

- **kb** (KnowledgeBase)
- **lmdb_path** (*str | None*)

**Return type**
None

increase_lmdb_map_size()

Increase the LMDB map size.

Note that the old database handle *may* no longer be valid after a map size increase. rhelmot could reproduce the error "Database handle belongs to another environment." in nix + CPython 3.13.13. Reopening all databases after increasing LMDB map size solves this issue.

**Return type**
None

reopen_lmdb()

Reopen the existing LMDB environment and all open databases in self._dbs.

reopen_lmdb_databases()

open_db(*db_name*, *unique=True*)

**Return type**
str

**Parameters**

- **db_name** (*str*)
- **unique** (*bool*)

begin_txn(*db_name*, *write=False*)

**Parameters**

- **db_name** (*str*)
- **write** (*bool*)

drop_db(*db_name*)

**Return type**
None

**Parameters**
**db_name** (*str*)

**cleanup**()

Close the LMDB environment and remove the rtdb directory if this is the last user of it. This method may be invoked multiple times (e.g., once by the atexit hook and once more by __del__) and should not fail.

**class** angr.knowledge_plugins.**StructuredCodeManager**

Bases: *KnowledgeBasePlugin*

A knowledge base plugin to store structured code generator results.

**__init__**(*kb*, *cache_limit=None*)

**Parameters**
**cache_limit** (*int | None*)

**get**(*key*, *default=None*)

**discard**(*key*)

**available_flavors**(*item*)

**all_flavors**(*item*)

**copy**()

**class** angr.knowledge_plugins.**TypesStore**

Bases: *KnowledgeBasePlugin*, UserDict[str, *TypeRef*]

A kb plugin that stores a mapping from name to TypeRef. It will return types from angr.sim_type.ALL_TYPES as a default.

**__init__**(*kb*)

**copy**()

**get_own**(*item*)

**Return type**
*TypeRef*

**Parameters**
**item** (*str*)

**iter_own**()

Iterate over all the names which are stored in this object - i.e. values() without ALL_TYPES

**iter_own_keys**()

**rename**(*old*, *new*)

**Parameters**

- **old** (*str*)
- **new** (*str*)

**unique_type_name**()

**Return type**
str

class angr.knowledge_plugins.**VariableManager**

Bases: *KnowledgeBasePlugin*

Manage variables.

**__init__**(*kb*)

**function_managers**: dict[int, *VariableManagerInternal*] | *SpillingVariableInternalDict*

**has_function_manager**(*key*)

**Return type**
bool

**Parameters**
**key** (*int*)

**get_function_manager**(*func_addr*)

**Return type**
*VariableManagerInternal*

**initialize_variable_names**()

**Return type**
None

**get_variable_accesses**(*variable*, *same_name=False*)

Get a list of all references to the given variable.

**Parameters**

- **variable** (*SimVariable*) – The variable.
- **same_name** (bool) – Whether to include all variables with the same variable name, or just based on the variable identifier.

**Return type**
list[*VariableAccess*]

**Returns**
All references to the variable.

**copy**()

**static convert_variable_list**(*vlist*, *manager*)

**Parameters**

- **vlist** (*list[Variable]*)
- **manager** (VariableManagerInternal)

**load_from_dwarf**(*cu_list=None*)

**Parameters**
**cu_list** (*list[CompilationUnit] | None*)

class angr.knowledge_plugins.**XRefManager**

Bases: *KnowledgeBasePlugin*, *Serializable*

**__init__**(*kb*)

**copy**()

**clear**()

**add_xref**(*xref*)

**add_xrefs**(*xrefs*)

**get_xrefs_by_ins_addr**(*ins_addr*)

**get_xrefs_by_dst**(*dst*)

**get_xrefs_by_dst_region**(*start*, *end*)

Get a set of XRef objects that point to a given address region bounded by start and end. Will only return absolute xrefs, not relative ones (like SP offsets)

**get_next_xref_addr_by_dst**(*addr*)

Get the next XRef object whose address is greater than or equal to the given address. Will only return absolute xrefs, not relative ones (like SP offsets)

**Return type**
int | None

**Parameters**
**addr** (*int*)

**get_xrefs_by_ins_addr_region**(*start*, *end*)

Get a set of XRef objects that originate at a given address region bounded by start and end. Useful for finding references from a basic block or function.

**Return type**
set[*XRef*]

## Submodules

| |
|---|
| *bookmarks* |
| *callsite_prototypes* |
| *cfg* |
| *comments* |
| *custom_strings* |
| *data* |
| *debug_variables* |
| *functions* |
| *indirect_jumps* |
| *key_definitions* |
| *labels* |
| *obfuscations* |
| *patches* |
| *plugin* |
| *propagations* |
| *rtdb* |
| *structured_code* |
| *types* |
| *variables* |
| *xrefs* |

### 10.22.1 angr.knowledge_plugins.bookmarks

**class** angr.knowledge_plugins.bookmarks.**Bookmark**

Bases: object

A user-marked address with an optional label.

**__init__**(*addr*, *label=''*, *created_at=None*)

**Parameters**

- **addr** (*int*)
- **label** (*str*)
- **created_at** (*float | None*)

**Return type**
None

**addr**

**label**

**created_at**

**class** angr.knowledge_plugins.bookmarks.**Bookmarks**

Bases: *KnowledgeBasePlugin*, list

Tracks bookmarks as a list of *Bookmark*, in creation order.

**get_bookmark**(*addr*)

**Return type**
*Bookmark* | None

**Parameters**
**addr** (*int*)

**has_bookmark**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

**add_bookmark**(*addr*, *label=''*)

Bookmark `addr`, or relabel the bookmark already there if `label` is non-empty. Returns the bookmark and whether it was newly created.

**Return type**
tuple[*Bookmark*, bool]

**Parameters**

- **addr** (*int*)
- **label** (*str*)

**remove_bookmark**(*bookmark*)

Remove `bookmark` if present. Returns whether it was removed.

**Return type**
bool

**Parameters**
**bookmark** (Bookmark)

**toggle_bookmark**(*addr*, *label=''*)

Add a bookmark at addr, or remove the one already there. Returns the new one, if any.

**Return type**
*Bookmark* | None

**Parameters**

- **addr** (*int*)
- **label** (*str*)

**sorted_bookmarks**()

**Return type**
list[*Bookmark*]

**next_bookmark**(*after_addr*)

The next bookmark by address, wrapping around.

**Return type**
*Bookmark* | None

**Parameters**
**after_addr** (*int | None*)

### 10.22.2 angr.knowledge_plugins.callsite_prototypes

**class** angr.knowledge_plugins.callsite_prototypes.**CallsitePrototypeKind**

Bases: Enum

Describes the type of a callsite prototype.

**INFERRED = 1**

**PROPAGATED = 2**

**MANUAL = 3**

**class** angr.knowledge_plugins.callsite_prototypes.**CallsitePrototypes**

Bases: *KnowledgeBasePlugin*

CallsitePrototypes manages callee prototypes at call sites.

**__init__**(*kb*)

**set_prototype**(*callsite_block_addr*, *cc*, *prototype*, ***, *kind=CallsitePrototypeKind.INFERRED*, *manual=False*, *propagated=False*)

**Return type**
None

**Parameters**

- **callsite_block_addr** (*int*)
- **cc** (SimCC)
- **prototype** (SimTypeFunction)
- **kind** (CallsitePrototypeKind)

- **manual** (*bool*)
- **propagated** (*bool*)

**get_cc**(*callsite_block_addr*, ***, *kind=CallsitePrototypeKind.INFERRED*)

**Return type**
*SimCC* | None

**Parameters**

- **callsite_block_addr** (*int*)
- **kind** (CallsitePrototypeKind)

**get_prototype**(*callsite_block_addr*, ***, *kind=CallsitePrototypeKind.INFERRED*)

**Return type**
*SimTypeFunction* | None

**Parameters**

- **callsite_block_addr** (*int*)
- **kind** (CallsitePrototypeKind)

**is_prototype_manual**(*callsite_block_addr*)

**Return type**
bool | None

**Parameters**
**callsite_block_addr** (*int*)

**is_prototype_certain**(*callsite_block_addr*)

**Return type**
bool | None

**Parameters**
**callsite_block_addr** (*int*)

**has_some_prototype**(*callsite_block_addr*)

**Return type**
bool

**Parameters**
**callsite_block_addr** (*int*)

**has_prototype**(*callsite_block_addr*, ***, *kind=CallsitePrototypeKind.INFERRED*)

**Return type**
bool

**Parameters**

- **callsite_block_addr** (*int*)
- **kind** (CallsitePrototypeKind)

**copy**()

### 10.22.3 angr.knowledge_plugins.cfg

**class** angr.knowledge_plugins.cfg.**BlockID**

Bases: object

A context-sensitive key for a SimRun object.

**__init__**(*addr*, *callsite_tuples*, *jump_type*)

**Parameters**

- **addr** (*int*)
- **callsite_tuples** (*tuple[int, ...] | None*)
- **jump_type** (*str*)

**callsite_repr**()

**static new**(*addr*, *callstack_suffix*, *jumpkind*)

**property func_addr**

**class** angr.knowledge_plugins.cfg.**CFGENode**

Bases: *CFGNode*

The CFGNode that is used in CFGEmulated.

**input_state**

**looping_times**

**depth**

**creation_failure_info**

**final_states**

**syscall**

**return_target**

**property callstack_key**

**property creation_failed**

**property syscall_name: str | None**

**downsize**()

Drop saved states.

**copy**()

**class** angr.knowledge_plugins.cfg.**CFGManager**

Bases: *KnowledgeBasePlugin*

This is the CFG manager, it manages CFGs

**__init__**(*kb*)

**new_model**(*prefix*, *addr_type='int'*)

**Parameters**
**addr_type** (*Literal['int', 'block_id', 'soot']*)

**copy**()

**get_most_accurate**()

**Return type**
*CFGModel* | None

**Returns**
The most accurate CFG present in the CFGManager, or None if it does not hold any.

**class** angr.knowledge_plugins.cfg.**CFGModel**

Bases: *Serializable*

This class describes a Control Flow Graph for a specific range of code.

**__init__**(*ident*, *cfg_manager=None*, *is_arm=False*, *cache_limit=None*, *db_batch_size=800*, *edge_cache_limit=None*, *edge_db_batch_size=800*, *addr_type='int'*)

**Parameters**

- **cache_limit** (*int | None*)
- **db_batch_size** (*int*)
- **edge_cache_limit** (*int | None*)
- **edge_db_batch_size** (*int*)
- **addr_type** (*Literal['int', 'block_id', 'soot']*)

**ident**

**is_arm**

**graph**: *SpillingCFG*

**jump_tables**: dict[int, *IndirectJump*]

**memory_data**: SortedDict[int, *MemoryData*]

**insn_addr_to_memory_data**: dict[int, *MemoryData*]

**normalized**

**edges_to_repair**

**property addr_type**: Literal['int', 'block_id', 'soot']

**property project**

**property node_addrs**: SortedList[int]

**nodes_by_addr**(*addr*)

**Return type**
Iterator[*CFGNode*]

**Parameters**
**addr** (*int*)

has_node_addr(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

mark_node_addr_has_return(*node_addr*, *has_return=True*)

**Return type**
None

**Parameters**

- **node_addr** (*int | SootAddressDescriptor*)
- **has_return** (*bool*)

node_addr_has_return(*node_addr*)

**Return type**
bool

**Parameters**
**node_addr** (*int | SootAddressDescriptor*)

copy()

add_node(*block_id*, *node*)

**Return type**
None

**Parameters**

- **block_id** (*int*)
- **node** (CFGNode)

remove_node(*block_id*, *node*)

Remove the given CFGNode instance. Note that this method *does not* remove the node from the graph.

**Parameters**

- **block_id** (int) – The Unique ID of the CFGNode.
- **node** (*CFGNode*) – The CFGNode instance to remove.

**Return type**
None

**Returns**
None

has_node_id(*node_id*)

**Return type**
bool

get_node(*block_id*)

Get a single node from Block ID.

**Parameters**
**block_id** (BlockID) – Block ID of the node.

**Return type**
*CFGNode* | None

**Returns**
The CFGNode, or None if the node does not exist.

**get_any_node**(*addr*, *is_syscall=None*, *anyaddr=False*, *force_fastpath=False*)

Get an arbitrary CFGNode (without considering their contexts) from our graph.

**Parameters**

- **addr** (int) – Address of the beginning of the basic block. Set anyaddr to True to support arbitrary address.
- **is_syscall** (bool | None) – Whether you want to get the syscall node or any other node. This is due to the fact that syscall SimProcedures have the same address as the target it returns to. None means get either, True means get a syscall node, False means get something that isn't a syscall node.
- **anyaddr** (bool) – If anyaddr is True, then addr doesn't have to be the beginning address of a basic block. By default the entire graph.nodes() will be iterated, and the first node containing the specific address is returned, which can be slow.
- **force_fastpath** (bool) – If force_fastpath is True, it will only perform a dict lookup in the graph._keys_by_addr dict.

**Return type**
*CFGNode* | None

**Returns**
A CFGNode if there is any that satisfies given conditions, or None otherwise

**get_all_nodes**(*addr*, *is_syscall=None*, *anyaddr=False*)

Get all CFGNodes whose address is the specified one.

**Parameters**

- **addr** (int) – Address of the node
- **is_syscall** (bool | None) – True returns the syscall node, False returns the normal CFGNode, None returns both
- **anyaddr** (*bool*)

**Return type**
list[*CFGNode*]

**Returns**
all CFGNodes

**get_all_nodes_intersecting_region**(*addr*, *size=1*)

Get all CFGNodes that intersect the given region.

**Parameters**

- **addr** (int) – Minimum address of target region.
- **size** (int) – Size of region, in bytes.

**Return type**
set[*CFGNode*]

`floor_addr`(*addr*)

Get the largest address that is less than or equal to the given address and has a CFGNode.

**Parameters**
**addr** (`int`) – The address to floor.

**Return type**
`int` | `None`

**Returns**
The largest address that is less than or equal to the given address and has a CFGNode, or None if no such address exists.

`ceil_addr`(*addr*)

Get the smallest address that is greater than or equal to the given address and has a CFGNode.

**Parameters**
**addr** (`int`) – The address to ceil.

**Return type**
`int` | `None`

**Returns**
The smallest address that is greater than or equal to the given address and has a CFGNode, or None if no such address exists.

`nodes`()

An iterator of all nodes in the graph.

**Returns**
The iterator.

**Return type**
iterator

`get_predecessors`(*cfgnode*, *excluding_fakeret=True*, *jumpkind=None*)

Get predecessors of a node in the control flow graph.

**Parameters**

- **cfgnode** (*CFGNode*) – The node.
- **excluding_fakeret** (`bool`) – True if you want to exclude all predecessors that is connected to the node with a fakeret edge.
- **jumpkind** (`str` | `None`) – Only return predecessors with the specified jumpkind. This argument will be ignored if set to None.

**Return type**
`list`[*CFGNode*]

**Returns**
A list of predecessors

`get_successors`(*node*, *excluding_fakeret=True*, *jumpkind=None*)

Get successors of a node in the control flow graph.

**Parameters**

- **node** (*CFGNode*) – The node.
- **excluding_fakeret** (`bool`) – True if you want to exclude all successors that is connected to the node with a fakeret edge.

- **jumpkind** (str | None) – Only return successors with the specified jumpkind. This argument will be ignored if set to None.

**Returns**
A list of successors

**Return type**
list[*CFGNode*]

**get_successors_and_jumpkinds**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the successor of the CFG node and the second element is the jumpkind of the successor.

**Parameters**

- **node** (CFGNode) – The node.
- **excluding_fakeret** (*bool*) – True if you want to exclude all successors that are fall-through successors.

**Returns**
A list of successors and their corresponding jumpkinds.

**Return type**
list[tuple[*CFGNode*, str]]

**get_successors_and_jumpkind**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the successor of the CFG node and the second element is the jumpkind of the successor.

**Parameters**

- **node** (CFGNode) – The node.
- **excluding_fakeret** (*bool*) – True if you want to exclude all successors that are fall-through successors.

**Returns**
A list of successors and their corresponding jumpkinds.

**Return type**
list[tuple[*CFGNode*, str]]

**get_predecessors_and_jumpkinds**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the predecessor of the CFG node and the second element is the jumpkind of the predecessor.

**Parameters**

- **node** (*CFGNode*) – The node.
- **excluding_fakeret** (bool) – True if you want to exclude all predecessors that are fall-through predecessors.

**Return type**
list[tuple[*CFGNode*, str]]

**Returns**
A list of predecessors and their corresponding jumpkinds.

**get_predecessors_and_jumpkind**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the predecessor of the CFG node and the second element is the jumpkind of the predecessor.

**Parameters**

- **node** (*CFGNode*) – The node.
- **excluding_fakeret** (bool) – True if you want to exclude all predecessors that are fall-through predecessors.

**Return type**
list[tuple[*CFGNode*, str]]

**Returns**
A list of predecessors and their corresponding jumpkinds.

**get_all_predecessors**(*cfgnode*, *depth_limit=None*)

Get all predecessors of a specific node on the control flow graph.

**Parameters**

- **cfgnode** (CFGNode) – The CFGNode object
- **depth_limit** (*int*) – Optional depth limit for the depth-first search

**Returns**
A list of predecessors in the CFG

**Return type**
list

**get_all_successors**(*cfgnode*, *depth_limit=None*)

Get all successors of a specific node on the control flow graph.

**Parameters**

- **cfgnode** (CFGNode) – The CFGNode object
- **depth_limit** (*int*) – Optional depth limit for the depth-first search

**Returns**
A list of successors in the CFG

**Return type**
list

**get_branching_nodes**()

Returns all nodes that has an out degree >= 2

**get_exit_stmt_idx**(*src_block*, *dst_block*)

Get the corresponding exit statement ID for control flow to reach destination block from source block. The exit statement ID was put on the edge when creating the CFG. Note that there must be a direct edge between the two blocks, otherwise an exception will be raised.

**Returns**
The exit statement ID

**add_memory_data**(*data_addr*, *data_type*, *data_size=None*)

Add a MemoryData entry to self.memory_data.

**Parameters**

- **data_addr** (int) – Address of the data
- **data_type** (*MemoryDataSort* | None) – Type of the memory data
- **data_size** (int | None) – Size of the memory data, or None if unknown for now.

**Return type**
bool

**Returns**
True if a new memory data entry is added, False otherwise.

**tidy_data_references**(*memory_data_addrs=None*, *exec_mem_regions=None*, *xrefs=None*, *seg_list=None*, *data_type_guessing_handlers=None*, *fill_gaps=True*, *new_mem_data_addrs=None*)

Go through all data references (or the ones as specified by memory_data_addrs) and determine their sizes and types if possible.

**Parameters**

- **memory_data_addrs** (list[int] | None) – A list of addresses of memory data, or None if tidying all known memory data entries.
- **exec_mem_regions** (list[tuple[int, int]] | None) – A list of start and end addresses of executable memory regions.
- **seg_list** (SegmentList | None) – The segment list that CFGFast uses during CFG recovery.
- **data_type_guessing_handlers** (list[Callable] | None) – A list of Python functions that will guess data types. They will be called in sequence to determine data types for memory data whose type is unknown.
- **fill_gaps** (bool) – If True, when a memory data entry is found to have a gap between its end and the next data entry, a new memory data entry will be created to fill the gap. fill_gaps should only be set to True at the end of CFG recovery when traversing the entire memory_data dict for the last time.
- **xrefs** (*XRefManager | None*)
- **new_mem_data_addrs** (*set[int] | None*)

**Return type**
bool

**Returns**
True if new data entries are found, False otherwise.

**remove_node_and_graph_node**(*node*)

Like *remove_node*, but also removes node from the graph.

**Parameters**
**node** (*CFGNode*) – The node to remove.

**Return type**
None

**get_intersecting_functions**(*addr*, *size=1*, *kb=None*)

Find all functions with nodes intersecting [addr, addr + size).

**Parameters**

- **addr** (int) – Minimum address of target region.
- **size** (int) – Size of region, in bytes.
- **kb** (*KnowledgeBase* | None) – Knowledge base to search for functions in.

**Return type**
set[*Function*]

**find_function_for_reflow_into_addr**(*addr*, *kb=None*)

Look for a function that flows into a new node at addr.

**Parameters**

- **addr** (int) – Address of new block.
- **kb** (*KnowledgeBase* | None) – Knowledge base to search for functions in.

**Return type**
*Function* | None

**clear_region_for_reflow**(*addr*, *size=1*, *kb=None*)

Remove nodes in the graph intersecting region [addr, addr + size).

Any functions that intersect the range, and their associated nodes in the CFG, will also be removed from the knowledge base for analysis.

**Parameters**

- **addr** (int) – Minimum address of target region.
- **size** (int) – Size of the region, in bytes.
- **kb** (*KnowledgeBase* | None) – Knowledge base to search for functions in.

**Return type**
None

**class** angr.knowledge_plugins.cfg.**CFGNode**

Bases: *Serializable*

This class stands for each single node in CFG.

**__init__**(*addr*, *size*, *cfg*, *simprocedure_name=None*, *no_ret=False*, *function_address=None*, *block_id=None*, *irsb=None*, *soot_block=None*, *instruction_addrs=None*, *thumb=False*, *byte_string=None*, *is_syscall=None*, *name=None*)

Note: simprocedure_name is not used to recreate the SimProcedure object. It's only there for better __repr__.

**Parameters**
**block_id** (BlockID | *int* | *None*)

**simprocedure_name**

**instruction_addrs**

**irsb**

**soot_block**

**property dirty**

**property function_address**

**property addr**

**property block_id**

**property byte_string: *bytes* | None**

**property is_syscall: bool**

property thumb: bool

property size: int

property no_ret: bool

property name

property successors

property predecessors

successors_and_jumpkinds(*excluding_fakeret=True*)

predecessors_and_jumpkinds(*excluding_fakeret=True*)

get_data_references(*kb=None*)

Get the known data references for this CFGNode via the knowledge base.

**Parameters**
**kb** – Which knowledge base to use; uses the global KB by default if none is provided

**Returns**
Generator yielding xrefs to this CFGNode's block.

**Return type**
iter

property accessed_data_references

Property providing a view of all the known data references for this CFGNode via the global knowledge base

**Returns**
Generator yielding xrefs to this CFGNode's block.

**Return type**
iter

property is_simprocedure

property callstack_key

copy()

merge(*other*)

Merges this node with the other, returning a new node that spans the both.

to_codenode()

property block: *Block* | *SootBlock* | None

class angr.knowledge_plugins.cfg.**IndirectJump**

Bases: *Serializable*

Describes an indirect jump or call site.

__init__(*addr*, *ins_addr*, *func_addr*, *jumpkind*, *stmt_idx*, *resolved_targets=None*, *jumptable=False*, *jumptable_addr=None*, *jumptable_size=None*, *jumptable_entry_size=None*, *jumptable_entries=None*, *jumptable_entries_guessed=False*, *type_=255*)

**Parameters**

- **addr** (*int*)

- **ins_addr** (*int*)
- **func_addr** (*int*)
- **jumpkind** (*str*)
- **stmt_idx** (*int*)
- **resolved_targets** (*list[int] | None*)
- **jumptable** (*bool*)
- **jumptable_addr** (*int | None*)
- **jumptable_size** (*int | None*)
- **jumptable_entry_size** (*int | None*)
- **jumptable_entries** (*list[int] | None*)
- **jumptable_entries_guessed** (*bool*)
- **type_** (*int | None*)

**addr**

**ins_addr**

**func_addr**

**jumpkind**

**stmt_idx**

**resolved_targets**

**jumptable**

**jumptables:** list[*JumptableInfo*]

**type**

**add_jumptable**(*addr*, *size*, *entry_size*, *entries*, *is_primary=False*, ***, *entries_guessed=False*)

**Return type**
None

**Parameters**

- **addr** (*int | None*)
- **size** (*int*)
- **entry_size** (*int*)
- **entries** (*list[int]*)
- **is_primary** (*bool*)
- **entries_guessed** (*bool*)

**property jumptable_addr:** int | None

**property jumptable_size:** int | None

**property jumptable_entry_size:** int | None

property jumptable_entries: list[int] | None

property jumptable_entries_guessed: bool | None

class angr.knowledge_plugins.cfg.IndirectJumpType

Bases: object

The type of an indirect jump or call.

Jumptable_AddressLoadedFromMemory = 0

Jumptable_AddressComputed = 1

Vtable = 3

Unknown = 255

class angr.knowledge_plugins.cfg.MemoryData

Bases: *Serializable*

MemoryData describes the syntactic content of a single address of memory.

*reference_size* reflects the size of *content*. It can be different from *size*, which is the actual size of the memory data item in memory. The intended way to get the actual content in memory is *self.content[:self.size]*.

__init__(*address*, *size*, *sort*, *pointer_addr=None*, *max_size=None*, *reference_size=None*)

**Parameters**

- **address** (*int*)
- **size** (*int*)
- **sort** (*str | None*)
- **pointer_addr** (*int | None*)
- **max_size** (*int | None*)
- **reference_size** (*int | None*)

addr: int

size: int

reference_size: int

sort: str | None

max_size: int | None

pointer_addr: int | None

content: *bytes* | None

property address

copy()

Make a copy of the MemoryData.

**Returns**

A copy of the MemoryData instance.

**Return type**
*MemoryData*

`fill_content`(*loader*)

Load data to fill self.content.

**Parameters**
**loader** – The project loader.

**Returns**
None

`class angr.knowledge_plugins.cfg.MemoryDataSort`

Bases: `object`

`Unspecified = None`

`Unknown = 'unknown'`

`Integer = 'integer'`

`PointerArray = 'pointer-array'`

`String = 'string'`

`UnicodeString = 'unicode'`

`SegmentBoundary = 'segment-boundary'`

`CodeReference = 'code reference'`

`GOTPLTEntry = 'GOT PLT Entry'`

`ELFHeader = 'elf-header'`

`FloatingPoint = 'fp'`

`Alignment = 'alignment'`

`PEImportDirectory = 'pe-import-directory'`

`PEExportDirectory = 'pe-export-directory'`

`PEDelayImportDirectory = 'pe-delay-import-directory'`

`EHFuncInfo = 'eh-funcinfo'`

`EHUnwindMapEntry = 'eh-unwindmapentry'`

`EH4ScopeTable = 'eh4-scopetable'`

`EHTryBlockMap = 'eh-tryblockmap'`

`EHHandlerType = 'eh-handlertype'`

continues on next page

Table 63 – continued from previous page

### Submodules

| | |
|---|---|
| *block_id* | |
| *cfg_manager* | |
| *cfg_model* | |
| *cfg_node* | |
| *indirect_jump* | |
| *memory_data* | |
| *spilling_cfg* | Spilling CFG Graph implementation with LRU caching and LMDB persistence. |
| *spilling_digraph* | SpillingDiGraph - a networkx.DiGraph subclass with LMDB-backed edge spilling. |
| *types* | |

### angr.knowledge_plugins.cfg.block_id

**class** angr.knowledge_plugins.cfg.block_id.**BlockID**

Bases: object

A context-sensitive key for a SimRun object.

**__init__**(*addr*, *callsite_tuples*, *jump_type*)

**Parameters**

- **addr** (*int*)
- **callsite_tuples** (*tuple[int, ...] | None*)
- **jump_type** (*str*)

**callsite_repr**()

**static new**(*addr*, *callstack_suffix*, *jumpkind*)

**property func_addr**

### angr.knowledge_plugins.cfg.cfg_manager

**class** angr.knowledge_plugins.cfg.cfg_manager.**CFGManager**

Bases: *KnowledgeBasePlugin*

This is the CFG manager, it manages CFGs

**__init__**(*kb*)

**new_model**(*prefix*, *addr_type='int'*)

**Parameters**

**addr_type** (*Literal['int', 'block_id', 'soot']*)

**copy**()

get_most_accurate()

**Return type**
*CFGModel* | None

**Returns**
The most accurate CFG present in the CFGManager, or None if it does not hold any.

## angr.knowledge_plugins.cfg.cfg_model

class angr.knowledge_plugins.cfg.cfg_model.**CFGModel**

Bases: *Serializable*

This class describes a Control Flow Graph for a specific range of code.

__init__(*ident*, *cfg_manager=None*, *is_arm=False*, *cache_limit=None*, *db_batch_size=800*, *edge_cache_limit=None*, *edge_db_batch_size=800*, *addr_type='int'*)

**Parameters**

- **cache_limit** (*int | None*)
- **db_batch_size** (*int*)
- **edge_cache_limit** (*int | None*)
- **edge_db_batch_size** (*int*)
- **addr_type** (*Literal['int', 'block_id', 'soot']*)

ident

is_arm

graph: *SpillingCFG*

jump_tables: dict[int, *IndirectJump*]

memory_data: SortedDict[int, *MemoryData*]

insn_addr_to_memory_data: dict[int, *MemoryData*]

normalized

edges_to_repair

property addr_type: Literal['int', 'block_id', 'soot']

property project

property node_addrs: SortedList[int]

nodes_by_addr(*addr*)

**Return type**
Iterator[*CFGNode*]

**Parameters**
**addr** (*int*)

**has_node_addr**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

**mark_node_addr_has_return**(*node_addr*, *has_return=True*)

**Return type**
None

**Parameters**

- **node_addr** (*int | SootAddressDescriptor*)
- **has_return** (*bool*)

**node_addr_has_return**(*node_addr*)

**Return type**
bool

**Parameters**
**node_addr** (*int | SootAddressDescriptor*)

**copy**()

**add_node**(*block_id*, *node*)

**Return type**
None

**Parameters**

- **block_id** (*int*)
- **node** (CFGNode)

**remove_node**(*block_id*, *node*)

Remove the given CFGNode instance. Note that this method *does not* remove the node from the graph.

**Parameters**

- **block_id** (int) – The Unique ID of the CFGNode.
- **node** (*CFGNode*) – The CFGNode instance to remove.

**Return type**
None

**Returns**
None

**has_node_id**(*node_id*)

**Return type**
bool

**get_node**(*block_id*)

Get a single node from Block ID.

**Parameters**
**block_id** (BlockID) – Block ID of the node.

**Return type**
*CFGNode* | None

**Returns**
The CFGNode, or None if the node does not exist.

**get_any_node**(*addr*, *is_syscall=None*, *anyaddr=False*, *force_fastpath=False*)

Get an arbitrary CFGNode (without considering their contexts) from our graph.

**Parameters**

- **addr** (int) – Address of the beginning of the basic block. Set anyaddr to True to support arbitrary address.
- **is_syscall** (bool | None) – Whether you want to get the syscall node or any other node. This is due to the fact that syscall SimProcedures have the same address as the target it returns to. None means get either, True means get a syscall node, False means get something that isn't a syscall node.
- **anyaddr** (bool) – If anyaddr is True, then addr doesn't have to be the beginning address of a basic block. By default the entire graph.nodes() will be iterated, and the first node containing the specific address is returned, which can be slow.
- **force_fastpath** (bool) – If force_fastpath is True, it will only perform a dict lookup in the graph._keys_by_addr dict.

**Return type**
*CFGNode* | None

**Returns**
A CFGNode if there is any that satisfies given conditions, or None otherwise

**get_all_nodes**(*addr*, *is_syscall=None*, *anyaddr=False*)

Get all CFGNodes whose address is the specified one.

**Parameters**

- **addr** (int) – Address of the node
- **is_syscall** (bool | None) – True returns the syscall node, False returns the normal CFGNode, None returns both
- **anyaddr** (*bool*)

**Return type**
list[*CFGNode*]

**Returns**
all CFGNodes

**get_all_nodes_intersecting_region**(*addr*, *size=1*)

Get all CFGNodes that intersect the given region.

**Parameters**

- **addr** (int) – Minimum address of target region.
- **size** (int) – Size of region, in bytes.

**Return type**
set[*CFGNode*]

**floor_addr**(*addr*)

Get the largest address that is less than or equal to the given address and has a CFGNode.

**Parameters**
**addr** (int) – The address to floor.

**Return type**
int | None

**Returns**
The largest address that is less than or equal to the given address and has a CFGNode, or None if no such address exists.

**ceil_addr**(*addr*)

Get the smallest address that is greater than or equal to the given address and has a CFGNode.

**Parameters**
**addr** (int) – The address to ceil.

**Return type**
int | None

**Returns**
The smallest address that is greater than or equal to the given address and has a CFGNode, or None if no such address exists.

**nodes**()

An iterator of all nodes in the graph.

**Returns**
The iterator.

**Return type**
iterator

**get_predecessors**(*cfgnode*, *excluding_fakeret=True*, *jumpkind=None*)

Get predecessors of a node in the control flow graph.

**Parameters**

- **cfgnode** (*CFGNode*) – The node.
- **excluding_fakeret** (bool) – True if you want to exclude all predecessors that is connected to the node with a fakeret edge.
- **jumpkind** (str | None) – Only return predecessors with the specified jumpkind. This argument will be ignored if set to None.

**Return type**
list[*CFGNode*]

**Returns**
A list of predecessors

**get_successors**(*node*, *excluding_fakeret=True*, *jumpkind=None*)

Get successors of a node in the control flow graph.

**Parameters**

- **node** (*CFGNode*) – The node.
- **excluding_fakeret** (bool) – True if you want to exclude all successors that is connected to the node with a fakeret edge.

- **jumpkind** (str | None) – Only return successors with the specified jumpkind. This argument will be ignored if set to None.

**Returns**
A list of successors

**Return type**
list[*CFGNode*]

**get_successors_and_jumpkinds**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the successor of the CFG node and the second element is the jumpkind of the successor.

**Parameters**

- **node** (CFGNode) – The node.
- **excluding_fakeret** (*bool*) – True if you want to exclude all successors that are fall-through successors.

**Returns**
A list of successors and their corresponding jumpkinds.

**Return type**
list[tuple[*CFGNode*, str]]

**get_successors_and_jumpkind**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the successor of the CFG node and the second element is the jumpkind of the successor.

**Parameters**

- **node** (CFGNode) – The node.
- **excluding_fakeret** (*bool*) – True if you want to exclude all successors that are fall-through successors.

**Returns**
A list of successors and their corresponding jumpkinds.

**Return type**
list[tuple[*CFGNode*, str]]

**get_predecessors_and_jumpkinds**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the predecessor of the CFG node and the second element is the jumpkind of the predecessor.

**Parameters**

- **node** (*CFGNode*) – The node.
- **excluding_fakeret** (bool) – True if you want to exclude all predecessors that are fall-through predecessors.

**Return type**
list[tuple[*CFGNode*, str]]

**Returns**
A list of predecessors and their corresponding jumpkinds.

**get_predecessors_and_jumpkind**(*node*, *excluding_fakeret=True*)

Get a list of tuples where the first element is the predecessor of the CFG node and the second element is the jumpkind of the predecessor.

**Parameters**

- **node** (*CFGNode*) – The node.
- **excluding_fakeret** (bool) – True if you want to exclude all predecessors that are fall-through predecessors.

**Return type**
list[tuple[*CFGNode*, str]]

**Returns**
A list of predecessors and their corresponding jumpkinds.

**get_all_predecessors**(*cfgnode*, *depth_limit=None*)

Get all predecessors of a specific node on the control flow graph.

**Parameters**

- **cfgnode** (CFGNode) – The CFGNode object
- **depth_limit** (*int*) – Optional depth limit for the depth-first search

**Returns**
A list of predecessors in the CFG

**Return type**
list

**get_all_successors**(*cfgnode*, *depth_limit=None*)

Get all successors of a specific node on the control flow graph.

**Parameters**

- **cfgnode** (CFGNode) – The CFGNode object
- **depth_limit** (*int*) – Optional depth limit for the depth-first search

**Returns**
A list of successors in the CFG

**Return type**
list

**get_branching_nodes**()

Returns all nodes that has an out degree >= 2

**get_exit_stmt_idx**(*src_block*, *dst_block*)

Get the corresponding exit statement ID for control flow to reach destination block from source block. The exit statement ID was put on the edge when creating the CFG. Note that there must be a direct edge between the two blocks, otherwise an exception will be raised.

**Returns**
The exit statement ID

**add_memory_data**(*data_addr*, *data_type*, *data_size=None*)

Add a MemoryData entry to self.memory_data.

**Parameters**

- **data_addr** (int) – Address of the data
- **data_type** (*MemoryDataSort* | None) – Type of the memory data
- **data_size** (int | None) – Size of the memory data, or None if unknown for now.

**Return type**
bool

**Returns**
True if a new memory data entry is added, False otherwise.

**tidy_data_references**(*memory_data_addrs=None*, *exec_mem_regions=None*, *xrefs=None*, *seg_list=None*, *data_type_guessing_handlers=None*, *fill_gaps=True*, *new_mem_data_addrs=None*)

Go through all data references (or the ones as specified by memory_data_addrs) and determine their sizes and types if possible.

**Parameters**

- **memory_data_addrs** (list[int] | None) – A list of addresses of memory data, or None if tidying all known memory data entries.
- **exec_mem_regions** (list[tuple[int, int]] | None) – A list of start and end addresses of executable memory regions.
- **seg_list** (SegmentList | None) – The segment list that CFGFast uses during CFG recovery.
- **data_type_guessing_handlers** (list[Callable] | None) – A list of Python functions that will guess data types. They will be called in sequence to determine data types for memory data whose type is unknown.
- **fill_gaps** (bool) – If True, when a memory data entry is found to have a gap between its end and the next data entry, a new memory data entry will be created to fill the gap. fill_gaps should only be set to True at the end of CFG recovery when traversing the entire memory_data dict for the last time.
- **xrefs** (XRefManager | *None*)
- **new_mem_data_addrs** (*set[int] | None*)

**Return type**
bool

**Returns**
True if new data entries are found, False otherwise.

**remove_node_and_graph_node**(*node*)

Like *remove_node*, but also removes node from the graph.

**Parameters**
**node** (*CFGNode*) – The node to remove.

**Return type**
None

**get_intersecting_functions**(*addr*, *size=1*, *kb=None*)

Find all functions with nodes intersecting [addr, addr + size).

**Parameters**

- **addr** (int) – Minimum address of target region.
- **size** (int) – Size of region, in bytes.
- **kb** (*KnowledgeBase* | None) – Knowledge base to search for functions in.

**Return type**
set[*Function*]

**find_function_for_reflow_into_addr**(*addr*, *kb=None*)

Look for a function that flows into a new node at addr.

**Parameters**

- **addr** (int) – Address of new block.
- **kb** (*KnowledgeBase* | None) – Knowledge base to search for functions in.

**Return type**
*Function* | None

**clear_region_for_reflow**(*addr*, *size=1*, *kb=None*)

Remove nodes in the graph intersecting region [addr, addr + size).

Any functions that intersect the range, and their associated nodes in the CFG, will also be removed from the knowledge base for analysis.

**Parameters**

- **addr** (int) – Minimum address of target region.
- **size** (int) – Size of the region, in bytes.
- **kb** (*KnowledgeBase* | None) – Knowledge base to search for functions in.

**Return type**
None

### angr.knowledge_plugins.cfg.cfg_node

**class** angr.knowledge_plugins.cfg.cfg_node.**CFGNodeCreationFailure**

Bases: object

This class contains additional information for whenever creating a CFGNode failed. It includes a full traceback and the exception messages.

**__init__**(*exc_info=None*, *to_copy=None*)

**short_reason**

**long_reason**

**traceback**

**class** angr.knowledge_plugins.cfg.cfg_node.**CFGNode**

Bases: *Serializable*

This class stands for each single node in CFG.

**__init__**(*addr*, *size*, *cfg*, *simprocedure_name=None*, *no_ret=False*, *function_address=None*, *block_id=None*, *irsb=None*, *soot_block=None*, *instruction_addrs=None*, *thumb=False*, *byte_string=None*, *is_syscall=None*, *name=None*)

Note: simprocedure_name is not used to recreate the SimProcedure object. It's only there for better __repr__.

**Parameters**
**block_id** (BlockID | *int* | *None*)

**simprocedure_name**

**instruction_addrs**

irsb

soot_block

property dirty

property function_address

property addr

property block_id

property byte_string: *bytes* | None

property is_syscall: bool

property thumb: bool

property size: int

property no_ret: bool

property name

property successors

property predecessors

successors_and_jumpkinds(*excluding_fakeret=True*)

predecessors_and_jumpkinds(*excluding_fakeret=True*)

get_data_references(*kb=None*)

Get the known data references for this CFGNode via the knowledge base.

**Parameters**
**kb** – Which knowledge base to use; uses the global KB by default if none is provided

**Returns**
Generator yielding xrefs to this CFGNode's block.

**Return type**
iter

property accessed_data_references

Property providing a view of all the known data references for this CFGNode via the global knowledge base

**Returns**
Generator yielding xrefs to this CFGNode's block.

**Return type**
iter

property is_simprocedure

property callstack_key

copy()

merge(*other*)

Merges this node with the other, returning a new node that spans the both.

to_codenode()

property block: *Block* | *SootBlock* | None

class angr.knowledge_plugins.cfg.cfg_node.**CFGENode**

Bases: *CFGNode*

The CFGNode that is used in CFGEmulated.

input_state

looping_times

depth

creation_failure_info

final_states

syscall

return_target

property callstack_key

property creation_failed

property syscall_name: str | None

downsize()

Drop saved states.

copy()

### angr.knowledge_plugins.cfg.indirect_jump

class angr.knowledge_plugins.cfg.indirect_jump.**IndirectJumpType**

Bases: object

The type of an indirect jump or call.

Jumptable_AddressLoadedFromMemory = 0

Jumptable_AddressComputed = 1

Vtable = 3

Unknown = 255

class angr.knowledge_plugins.cfg.indirect_jump.**JumptableInfo**

Bases: object

Describes a jump table or a vtable.

__init__(*addr*, *size*, *entry_size*, *entries*, ***, *entries_guessed=False*)

**Parameters**

- **addr** (*int | None*)
- **size** (*int*)
- **entry_size** (*int*)

- **entries** (*list[int]*)
- **entries_guessed** (*bool*)

**addr**

**size**

**entry_size**

**entries**

**entries_guessed**

**serialize_to_cmessage**()

**classmethod parse_from_cmessage**(*cmsg*)

**Return type**
*JumptableInfo*

**class** angr.knowledge_plugins.cfg.indirect_jump.**IndirectJump**

Bases: *Serializable*

Describes an indirect jump or call site.

**__init__**(*addr*, *ins_addr*, *func_addr*, *jumpkind*, *stmt_idx*, *resolved_targets=None*, *jumptable=False*, *jumptable_addr=None*, *jumptable_size=None*, *jumptable_entry_size=None*, *jumptable_entries=None*, *jumptable_entries_guessed=False*, *type_=255*)

**Parameters**

- **addr** (*int*)
- **ins_addr** (*int*)
- **func_addr** (*int*)
- **jumpkind** (*str*)
- **stmt_idx** (*int*)
- **resolved_targets** (*list[int] | None*)
- **jumptable** (*bool*)
- **jumptable_addr** (*int | None*)
- **jumptable_size** (*int | None*)
- **jumptable_entry_size** (*int | None*)
- **jumptable_entries** (*list[int] | None*)
- **jumptable_entries_guessed** (*bool*)
- **type_** (*int | None*)

**addr**

**ins_addr**

**func_addr**

**jumpkind**

stmt_idx

resolved_targets

jumptable

jumptables: list[*JumptableInfo*]

type

add_jumptable(*addr*, *size*, *entry_size*, *entries*, *is_primary=False*, ***, *entries_guessed=False*)

**Return type**
None

**Parameters**

- **addr** (*int | None*)
- **size** (*int*)
- **entry_size** (*int*)
- **entries** (*list[int]*)
- **is_primary** (*bool*)
- **entries_guessed** (*bool*)

property jumptable_addr: int | None

property jumptable_size: int | None

property jumptable_entry_size: int | None

property jumptable_entries: list[int] | None

property jumptable_entries_guessed: bool | None

### angr.knowledge_plugins.cfg.memory_data

class angr.knowledge_plugins.cfg.memory_data.**MemoryDataSort**

Bases: object

Unspecified = None

Unknown = 'unknown'

Integer = 'integer'

PointerArray = 'pointer-array'

String = 'string'

UnicodeString = 'unicode'

SegmentBoundary = 'segment-boundary'

CodeReference = 'code reference'

GOTPLTEntry = 'GOT PLT Entry'

ELFHeader = 'elf-header'

FloatingPoint = 'fp'

Alignment = 'alignment'

PEImportDirectory = 'pe-import-directory'

PEExportDirectory = 'pe-export-directory'

PEDelayImportDirectory = 'pe-delay-import-directory'

EHFuncInfo = 'eh-funcinfo'

EHUnwindMapEntry = 'eh-unwindmapentry'

EH4ScopeTable = 'eh4-scopetable'

EHTryBlockMap = 'eh-tryblockmap'

EHHandlerType = 'eh-handlertype'

class angr.knowledge_plugins.cfg.memory_data.**MemoryData**

Bases: *Serializable*

MemoryData describes the syntactic content of a single address of memory.

*reference_size* reflects the size of *content*. It can be different from *size*, which is the actual size of the memory data item in memory. The intended way to get the actual content in memory is *self.content[:self.size]*.

**__init__**(*address*, *size*, *sort*, *pointer_addr=None*, *max_size=None*, *reference_size=None*)

**Parameters**

- **address** (*int*)
- **size** (*int*)
- **sort** (*str | None*)
- **pointer_addr** (*int | None*)
- **max_size** (*int | None*)
- **reference_size** (*int | None*)

**addr:** int

**size:** int

**reference_size:** int

**sort:** str | None

**max_size:** int | None

**pointer_addr:** int | None

**content:** *bytes* | None

property address

**copy**()

Make a copy of the MemoryData.

**Returns**
A copy of the MemoryData instance.

**Return type**
*MemoryData*

**fill_content**(*loader*)

Load data to fill self.content.

**Parameters**
**loader** – The project loader.

**Returns**
None

### angr.knowledge_plugins.cfg.spilling_cfg

Spilling CFG Graph implementation with LRU caching and LMDB persistence.

This module provides SpillingCFGNodeDict and SpillingCFGGraph classes that implement disk-backed storage for CFGNode instances, following the SpillingFunctionDict pattern.

**class** angr.knowledge_plugins.cfg.spilling_cfg.**SpillingCFGNodeDict**

Bases: object

A dict-like container for CFGNode instances with LRU caching and LMDB spilling.

This class keeps only the most recently accessed N nodes in memory, spilling others to an LMDB database on disk.

**Variables**

- **cache_limit** – The maximum number of nodes to keep in memory.
- **rtdb** (*RuntimeDb* | None) – A reference to the RuntimeDb knowledge base plugin.
- **_lru_order** (OrderedDict[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor, None]) – An OrderedDict tracking the eviction order of cached nodes.
- **_spilled_keys** (set[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor]) – A set of block_ids that have been spilled to LMDB.
- **_db_batch_size** (int) – The number of nodes that are evicted in a single batch.
- **_eviction_enabled** (bool) – A flag indicating whether eviction is currently enabled.

**__init__**(*rtdb*, *cfg_model=None*, *cache_limit=1000*, *db_batch_size=200*)

**Parameters**

- **rtdb** (RuntimeDb | *None*)
- **cfg_model** (CFGModel | *None*)
- **cache_limit** (*int*)
- **db_batch_size** (*int*)

**get**(*block_key*, *default=None*)

**Return type**
*CFGNode* | None

**Parameters**

- **block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)
- **default** (*CFGNode | None*)

**keys**()

**Return type**
Iterator[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor]

**values**()

**Return type**
Iterator[*CFGNode*]

**items**()

**Return type**
Iterator[tuple[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor, *CFGNode*]]

**clear**()

**Return type**
None

**copy**()

**Return type**
*SpillingCFGNodeDict*

**property cache_limit: int**

**property db_batch_size: int**

**property cached_count: int**

**property spilled_count: int**

**property total_count: int**

**is_cached**(*block_key*)

**Return type**
bool

**Parameters**
**block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

**bulk_import_serialized**(*items*)

Bulk-import already-serialized CFG nodes directly into the LMDB backing store and register them as spilled, without deserializing them. Each payload must be in the format that _save_to_lmdb() writes: a type byte (0x00 for CFGNode, 0x01 for CFGENode) followed by the serialized protobuf message.

**Parameters**
**items** (list[tuple[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor, bytes]]) – A list of (block key, serialized node payload) tuples.

**Return type**
None

**load_all_spilled**()

**Return type**
None

**evict_all_cached**()

**Return type**
None

angr.knowledge_plugins.cfg.spilling_cfg.**get_block_key**(*node*)

Get the unique identifier for a CFGNode. Typically this unique identifier contains the address of the block and the looping_times of the block (in case there are multiple blocks with the same address, which may happen after loop unrolling in a CFGEmulated instance).

**Overloads**

- **node** (CFGNode) → CFGNODE_K | SOOTNODE_K
- **node** (CFGENode) → CFGENODE_K

**Parameters**
**node** (*CFGNode* | *CFGENode*) – The CFGNode or CFGENode instance to get the block key for.

**Returns**
The unique identifier.

**Return type**
tuple[int, int] | tuple[BlockID, int, int] | *SootAddressDescriptor*

angr.knowledge_plugins.cfg.spilling_cfg.**block_key_to_addr**(*block_key*)

Extract the address from a block key.

**Overloads**

- **block_key** (CFGNODE_K | CFGENODE_K) → int
- **block_key** (SOOTNODE_K) → SootAddressDescriptor

**Parameters**
**block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

**Return type**
int | *SootAddressDescriptor*

angr.knowledge_plugins.cfg.spilling_cfg.**block_key_to_size**(*block_key*)

Extract the size from a block key, if present.

**Return type**
int | None

**Parameters**
**block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

class angr.knowledge_plugins.cfg.spilling_cfg.**SpillingCFG**

Bases: object

A graph wrapper that stores CFGNode instances in a spilling dict while keeping only primitive keys in the underlying networkx graph.

This provides a networkx-compatible interface while supporting disk-backed storage for large CFGs.

addr_type must be "int", "block_id", or "soot". You can change addr_type before the first node is inserted but not after, since it affects how keys are serialized and deserialized.

**__init__**(*rtdb=None*, *cfg_model=None*, *cache_limit=None*, *db_batch_size=800*, *edge_cache_limit=None*, *edge_db_batch_size=800*, *addr_type='int'*)

**Parameters**

- **rtdb** (RuntimeDb | *None*)
- **cfg_model** (CFGModel | *None*)
- **cache_limit** (*int* | *None*)
- **db_batch_size** (*int*)
- **edge_cache_limit** (*int* | *None*)
- **edge_db_batch_size** (*int*)
- **addr_type** (*Literal['int', 'block_id', 'soot']*)

**property addr_type:** str

**get_node_by_key**(*block_key*)

Get a CFGNode by block_id, with fallback to graph node data.

**Return type**
*CFGNode*

**Parameters**
**block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

**property node_keys:** Iterator[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor]

Get an iterator over all block_keys in the graph.

**add_node**(*node*, ***attr*)

**Return type**
None

**Parameters**
**node** (CFGNode)

**export_serialized_nodes**()

Export every node as the serialized bytes of its CFGNode/CFGENode protobuf message (without the LMDB type byte), e.g. for dumping into an angr database.

Nodes are byte-copied directly out of the LMDB backing store without being deserialized and re-serialized.

**Return type**
list[tuple[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor, bytes, bool]]

**Returns**
A list of (block key, serialized CFGNode message bytes, copied_from_lmdb) tuples.

**bulk_import_serialized_nodes**(*items*)

Bulk-import already-serialized nodes directly into the LMDB backing store as spilled entries, without constructing any CFGNode objects.

**Parameters**
**items** (list[tuple[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor, int | SootAddressDescriptor, bytes]]) – A list of (block key, node address, serialized node payload) tuples. See SpillingCFGNodeDict.bulk_import_serialized() for the payload format.

**Return type**
None

**remove_node**(*node*)

**Return type**
None

**Parameters**
**node** (CFGNode)

**has_node**(*node*)

**Return type**
bool

**Parameters**
**node** (CFGNode)

**nodes_by_addr**(*addr*)

**Return type**
Iterator[*CFGNode*]

**Parameters**
**addr** (*int*)

**has_node_addr**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

**number_of_nodes**()

**Return type**
int

**property nodes: _NodeView**

Return a view of nodes supporting len(), iteration, and call with data=True.

**add_edge**(*src*, *dst*, ***attr*)

**Return type**
None

**Parameters**

- **src** (CFGNode)
- **dst** (CFGNode)

**add_edge_by_key**(*src_block_key*, *dst_block_key*, ***attr*)

**Return type**
None

**Parameters**

- **src_block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)
- **dst_block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

**remove_edge**(*src*, *dst*)

**Return type**
None

**Parameters**

- **src** (CFGNode)
- **dst** (CFGNode)

**remove_edge_by_key**(*src_block_key*, *dst_block_key*)

**Return type**
None

**Parameters**

- **src_block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)
- **dst_block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

**has_edge**(*src*, *dst*)

**Return type**
bool

**Parameters**

- **src** (CFGNode)
- **dst** (CFGNode)

**has_edge_by_key**(*src_block_key*, *dst_block_key*)

**Return type**
bool

**Parameters**

- **src_block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)
- **dst_block_key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

**get_edge_data**(*src*, *dst*, *default=None*)

**Return type**
dict | None

**Parameters**

- **src** (CFGNode)
- **dst** (CFGNode)

**number_of_edges**()

**Return type**
int

**property edges: _EdgeView**

Return a view of edges supporting len(), iteration, and call with data=True.

**property call_destination_keys: set[tuple[int, int] | tuple[*BlockID*, int, int] | SootAddressDescriptor]**

Return the set of block keys that are destinations of call/syscall edges.

**call_destination_nodes**()

Yield CFGNode for each call/syscall destination.

**Return type**
Iterator[*CFGNode*]

**predecessors**(*node*)

**Return type**
Iterator[*CFGNode*]

**Parameters**
**node** (CFGNode)

**successors**(*node*)

**Return type**
Iterator[*CFGNode*]

**Parameters**
**node** (CFGNode)

**property in_edges: _InEdgeView**

Return a view of in-edges supporting call, subscript, len, and iteration.

**property out_edges: _OutEdgeView**

Return a view of out-edges supporting call, subscript, len, and iteration.

**property in_degree: _InDegreeView**

Return a view of in-degrees supporting call, subscript, len, and iteration.

**property out_degree: _OutDegreeView**

Return a view of out-degrees supporting call, subscript, len, and iteration.

**out_edges_by_key**(*key*, *data=False*)

**Overloads**

- **self**, **key** (K), **data** (Literal[False]) → Generator[tuple[K, K]]

- **self**, **key** (K), **data** (Literal[True]) → Generator[tuple[K, K, dict]]

**Parameters**

- **key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)
- **data** (*bool*)

**Return type**

*Generator*[tuple[tuple[int, int] | tuple[BlockID, int, int] | *SootAddressDescriptor*, tuple[int, int] | tuple[BlockID, int, int] | *SootAddressDescriptor*] | tuple[tuple[int, int] | tuple[BlockID, int, int] | *SootAddressDescriptor*, tuple[int, int] | tuple[BlockID, int, int] | *SootAddressDescriptor*, dict]]

**out_degree_by_key**(*key*)

**Return type**

int

**Parameters**

**key** (*tuple[int, int] | tuple[BlockID, int, int] | SootAddressDescriptor*)

**copy**()

**Return type**

*SpillingCFG*

**subgraph**(*nodes*)

Return a subgraph as a regular networkx DiGraph with CFGNode instances. This is useful for algorithms that need a pure networkx graph.

**Return type**

DiGraph

**to_networkx**()

Convert to a pure networkx DiGraph with CFGNode instances as nodes. Warning: This loads all spilled nodes into memory.

**Return type**

DiGraph

**from_networkx**(*nx_graph*)

Load graph structure from a networkx DiGraph with CFGNode instances as nodes.

**Return type**

None

**Parameters**

**nx_graph** (*DiGraph*)

**property cache_limit: int | None**

**property db_batch_size: int**

**property cached_count: int**

**property spilled_count: int**

**load_all_spilled**()

**Return type**
None

**evict_all_cached**()

**Return type**
None

**set_rtdb**(*rtdb*)

(Re-)attach a RuntimeDb to this graph and its spilling containers. This is used after unpickling (rtdb references are not preserved across pickling) so that node/edge spilling works again.

**Return type**
None

**Parameters**
**rtdb** (RuntimeDb | *None*)

### angr.knowledge_plugins.cfg.spilling_digraph

SpillingDiGraph - a networkx.DiGraph subclass with LMDB-backed edge spilling.

This module provides SpillingAdjDict and SpillingDiGraph classes that implement disk-backed storage for graph adjacency data, following the SpillingCFGNodeDict pattern.

Edge attributes are serialized using the Edge protobuf message from primitives.proto.

**class** angr.knowledge_plugins.cfg.spilling_digraph.**DirtyDict**

Bases: UserDict, Generic

A simple dict subclass that tracks whether it has been modified since creation or last reset.

This is used for edge attribute dicts to know when they need to be re-serialized and saved to LMDB.

**__init__**(**args*, *dirty=False*, ***kwargs*)

**Parameters**
**dirty** (*bool*)

**class** angr.knowledge_plugins.cfg.spilling_digraph.**SpillingAdjDict**

Bases: MutableMapping

A dict-like container for adjacency data with LRU caching and LMDB spilling.

Keys are node keys (block_key tuples), values are inner adjacency dicts (mapping neighbor_key -> edge_attr_dict).

When the number of cached entries exceeds `cache_limit + db_batch_size`, the `db_batch_size` least recently used entries are evicted to LMDB.

Edge attributes within each inner dict are serialized/deserialized using the `Edge` protobuf message from `primitives.proto`.

**__init__**(*addr_type*, *rtdb=None*, *cache_limit=1000*, *db_batch_size=800*)

**Parameters**

- **addr_type** (*Literal['int', 'block_id', 'soot']*)
- **rtdb** (RuntimeDb | *None*)
- **cache_limit** (*int*)

- **db_batch_size** (*int*)

**load_all_spilled**()

Load all spilled entries back into memory.

**Return type**
None

**evict_all_cached**()

Evict all cached entries to LMDB.

**Return type**
None

**copy**()

**Return type**
*SpillingAdjDict*

**class** angr.knowledge_plugins.cfg.spilling_digraph.**SpillingDiGraph**

Bases: DiGraph

A networkx DiGraph subclass whose `adjlist_outer_dict_factory` produces *SpillingAdjDict* instances, enabling LRU-based LMDB spilling of adjacency data (edges and their attributes).

**Parameters**

- **rtdb** (*RuntimeDb* | None) – RuntimeDb used for LMDB access.
- **edge_cache_limit** (int) – Maximum adjacency entries to keep in memory per outer dict (_adj and _pred each).
- **db_batch_size** (int) – Number of entries evicted in a single batch.

**property addr_type:** Literal['int', 'block_id', 'soot']

**load_all_spilled_edges**()

Load all spilled adjacency entries back into memory.

**Return type**
None

**set_rtdb**(*rtdb*)

(Re-)attach a RuntimeDb to this graph and its adjacency containers. This is used after unpickling (rtdb references are not preserved across pickling) so that edge spilling works again.

**Return type**
None

**Parameters**
**rtdb** (RuntimeDb | *None*)

**evict_all_cached_edges**()

Evict all cached adjacency entries to LMDB.

**Return type**
None

**set_edge_eviction_enabled**(*enabled*)

Enable or disable eviction of adjacency entries. Disabling eviction is useful during bulk edge insertion; call spill_down_edges() afterwards to bring the caches back within their limits.

**Return type**
None

**Parameters**
**enabled** (*bool*)

**spill_down_edges**()

Evict least-recently-used adjacency entries until the caches are back within their limits.

**Return type**
None

### angr.knowledge_plugins.cfg.types

angr.knowledge_plugins.cfg.types.**SOOTNODE_K**

alias of SootAddressDescriptor

## 10.22.4 angr.knowledge_plugins.comments

**class** angr.knowledge_plugins.comments.**CommentKind**

Bases: IntEnum

How a comment is displayed.

PLAIN shows only at its own address. FUNCTION renders as a header block above the function, which is also what the decompiler does with a comment on a function entry. REPEATABLE additionally shows at every location referencing the commented address.

**PLAIN = 0**

**FUNCTION = 1**

**REPEATABLE = 2**

**property display_name: str**

**__new__**(*value*)

**class** angr.knowledge_plugins.comments.**Comment**

Bases: object

A comment together with its display metadata. Rebuilt on demand from the knowledge base.

**__init__**(*addr*, *text*, *kind*, *func_addr=None*, *func_name=None*)

**Parameters**

- **addr** (*int*)
- **text** (*str*)
- **kind** (CommentKind)
- **func_addr** (*int | None*)
- **func_name** (*str | None*)

**Return type**
None

**addr**

**text**

**kind**

**func_addr**

**func_name**

**property first_line:** str

**class** angr.knowledge_plugins.comments.**Comments**

Bases: *KnowledgeBasePlugin*, dict

Tracks comments via a Dict of Address -> Text, along with a per-address *CommentKind*.

**property kinds:** dict[int, *CommentKind*]

Explicitly-set comment kinds. Addresses not present here use *kind_of()* defaults.

**kind_of**(*addr*)

**Return type**
*CommentKind*

**Parameters**
**addr** (*int*)

**set_kind**(*addr*, *kind*)

Set the display kind for the comment at addr, or reset it to the default with None.

**Return type**
None

**Parameters**

- **addr** (*int*)
- **kind** (CommentKind | *None*)

**is_function_entry**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

**function_comment**(*func_addr*)

The comment rendered as a function's header block, if any.

**Return type**
str | None

**Parameters**
**func_addr** (*int*)

**inline_comment**(*addr*)

The comment rendered next to the instruction at addr. A comment that already renders as a function header block is not repeated inline.

**Return type**
str | None

**Parameters**
**addr** (*int*)

`iter_comments()`

All comments in address order, with their kinds and owning functions.

**Return type**
Iterator[*Comment*]

`repeatable_comments_at(ins_addr)`

Repeatable comments that should show at `ins_addr` because it references their address.

**Return type**
list[tuple[int, str]]

**Parameters**
**ins_addr** (*int*)

`invalidate()`

Drop the cached repeatable-comment map, e.g. after xrefs or functions change.

**Return type**
None

### 10.22.5 angr.knowledge_plugins.custom_strings

`class angr.knowledge_plugins.custom_strings.CustomStrings`

Bases: *KnowledgeBasePlugin*

Store new strings that are recovered during various analysis. Each string has a unique ID associated.

`__init__(kb)`

`allocate(s)`

**Return type**
int

**Parameters**
**s** (bytes)

`copy()`

### 10.22.6 angr.knowledge_plugins.data

`class angr.knowledge_plugins.data.Data`

Bases: *KnowledgeBasePlugin*

The knowledge what purpose this plugin serves has been lost to the passing of time but the linter does not care for these failures of mere mortals and demands a docstring anyway. The pact has been made, and no violations of the rules will be tolerated, even if the spirit does not match the letter. Making the plugin smaller has only increased the weight of the failure, and thus this file has drawn its ire.

The only thing left to do is to attempt to find meaning in the meaninglessness, as the only act of rebellion against the uncaring forces that bind us. For is this not what being human is all about?

`copy()`

### 10.22.7 angr.knowledge_plugins.debug_variables

**class** angr.knowledge_plugins.debug_variables.**DebugVariableContainer**

Bases: object

Variable tree for variables with same name to lock up which variable is visible at a given program counter address.

**__init__**()

It is recommended to use DebugVariableManager.add_variable() instead

**from_pc**(*pc*)

Returns the visible variable (if any) for a given pc address.

**Return type**
Variable

**class** angr.knowledge_plugins.debug_variables.**DebugVariable**

Bases: *DebugVariableContainer*

**Variables**

- **low_pc** – Start of the visibility scope of the variable as program counter address (rebased)
- **high_pc** – End of the visibility scope of the variable as program counter address (rebased)
- **cle_variable** – Original variable from cle

**__init__**(*low_pc*, *high_pc*, *cle_variable*)

It is recommended to use DebugVariableManager.add_variable() instead

**Parameters**

- **low_pc** (*int*)
- **high_pc** (*int*)
- **cle_variable** (*Variable*)

**contains**(*dvar*)

**Return type**
bool

**Parameters**
**dvar** (DebugVariable)

**test_unsupported_overlap**(*dvar*)

Test for an unsupported overlapping

**Parameters**
**dvar** (*DebugVariable*) – Second DebugVariable to compare with

**Return type**
bool

**Returns**
True if there is an unsupported overlapping

**class** angr.knowledge_plugins.debug_variables.**DebugVariableManager**

Bases: *KnowledgeBasePlugin*

Structure to manage and access variables with different visibility scopes.

__init__(*kb*)

**Parameters**
**kb** (KnowledgeBase)

from_name_and_pc(*var_name*, *pc_addr*)

Get a variable from its string in the scope of pc.

**Return type**
Variable

**Parameters**

- **var_name** (*str*)
- **pc_addr** (*int*)

from_name(*var_name*)

Get the variable container for all variables named var_name

**Parameters**
**var_name** (str) – name for a variable

**Return type**
*DebugVariableContainer*

add_variable(*cle_var*, *low_pc*, *high_pc*)

Add/load a variable

**Parameters**

- **cle_variable** – The variable to add
- **low_pc** (int) – Start of the visibility scope of the variable as program counter address (rebased)
- **high_pc** (int) – End of the visibility scope of the variable as program counter address (rebased)
- **cle_var** (*Variable*)

add_variable_list(*vlist*, *low_pc*, *high_pc*)

Add all variables in a list with the same visibility range

**Parameters**

- **vlist** (list[Variable]) – A list of cle variables to add
- **low_pc** (int) – Start of the visibility scope as program counter address (rebased)
- **high_pc** (int) – End of the visibility scope as program counter address (rebased)

load_from_dwarf(*elf_object=None*, *cu=None*)

Automatically load all variables (global/local) from the DWARF debugging info

**Parameters**

- **elf_object** (ELF) – Optional, when only one elf object should be considered (e.g. p.loader.main_object)
- **cu** (CompilationUnit) – Optional, when only one compilation unit should be considered

### 10.22.8 angr.knowledge_plugins.functions

**class** angr.knowledge_plugins.functions.**Function**

Bases: *Serializable*

A representation of a function and various information about it.

**Variables**

- **meta_only** (bool) – Whether this function only contains meta-information and in read-only mode.
- **_dirty** (bool) – Whether this function has been modified since last serialization.
- **evicted** (bool) – Whether this function has been evicted from FunctionManager to external storage.

**__init__**(*function_manager*, *addr*, *name=None*, *syscall=None*, *is_simprocedure=None*, *binary_name=None*, *is_plt=None*, *returning=None*, *alignment=False*, *calling_convention=None*, *prototype=None*, *prototype_libname=None*, *prototype_source=None*, *is_prototype_guessed=True*)

Function constructor. If the optional parameters are not provided, they will be automatically determined upon the creation of a Function object.

**Parameters**

- **addr** (int) – The address of the function.
- **name** (*str*) – The name of the function.
- **syscall** (*bool*) – Whether this function is a syscall or not.
- **is_simprocedure** (bool | None) – Whether this function is a SimProcedure or not.
- **binary_name** (*str*) – Name of the binary where this function is.
- **is_plt** (bool | None) – If this function is a PLT entry.
- **returning** (*bool*) – If this function returns.
- **alignment** (*bool*) – If this function acts as an alignment filler. Such functions usually only contain nops.
- **function_manager** (FunctionManager | *None*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **prototype_libname** (*str* | *None*)
- **prototype_source** (PrototypeSource | *None*)
- **is_prototype_guessed** (*bool*)

**transition_graph**

**normalized**

**addr**

**startpoint**

**bp_on_stack**

**retaddr_on_stack**

**sp_delta**

**tags**

**ran_cca**

**meta_only:** *bool*

**evicted:** *bool*

**is_default_name**

**previous_names**

**binary_name**

**property name**

**property from_signature:** *str | None*

**property project:** *Project | None*

**property returning**

**property calling_convention:** *SimCC | None*

**property prototype:** *SimTypeFunction | None*

**property prototype_libname**

**property is_prototype_guessed:** *bool*

**property is_prototype_groundtruth:** *bool*

True if the prototype comes from outside of the decompiler (SimProcedures, signatures, or the user) and may be fed back into type inference as ground truth. Prototypes inferred by the decompiler itself are excluded so that re-decompiling a function does not freeze its own earlier guess.

**property prototype_source:** *PrototypeSource*

**property info:** *FunctionInfo*

**property is_plt:** *bool*

**property is_simprocedure:** *bool*

**property is_syscall:** *bool*

**property is_alignment:** *bool*

**property blocks**

An iterator of all local blocks in the current function.

**Returns**
angr.lifter.Block instances.

**property code_nodes:** *dict[int, CodeNode]*

property cyclomatic_complexity

The cyclomatic complexity of the function.

Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. It is computed using the formula: M = E - N + 2P, where E = the number of edges in the graph, N = the number of nodes in the graph, P = the number of connected components.

The cyclomatic complexity value is lazily computed and cached for future use. Initially this value is None until it is computed for the first time

**Returns**
The cyclomatic complexity of the function.

**Return type**
int

property xrefs: Iterator[*XRef*]

An iterator of all xrefs of the current function.

**Returns**
angr.knowledge_plugins.xrefs.xref.XRef instances.

property block_addrs

An iterator of all local block addresses in the current function.

**Returns**
block addresses.

property block_addrs_set

Return a set of block addresses for a better performance of inclusion tests.

**Returns**
A set of block addresses.

**Return type**
set

get_block(*addr*, *size=None*, *byte_string=None*)

Getting a block out of the current function.

**Parameters**

- **addr** (int) – The address of the block.
- **size** (int | None) – The size of the block. This is optional. If not provided, angr will load
- **byte_string** (bytes | None)

**Returns**

get_block_size(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*int*)

property nodes: Iterable[*CodeNode*]

**get_node**(*addr*)

**Return type**
*BlockNode* | None

**property has_unresolved_jumps**

**property has_unresolved_calls**

**property operations**
All of the operations that are done by this functions.

**property code_constants**
All of the constants that are used by this functions's code.

**classmethod parse_from_cmessage**(*cmsg*, ***kwargs*)

**Parameters**
**cmsg**

**Return Function**
The function instantiated out of the cmsg data.

**string_references**(*minimum_length=2*)
All of the constant string references used by this function.

**Parameters**
**minimum_length** – The minimum length of strings to find (default is 1)

**Returns**
A generator yielding tuples of (address, string) where is address is the location of the string in memory.

**property local_runtime_values**
Tries to find all runtime values of this function which do not come from inputs. These values are generated by starting from a blank state and reanalyzing the basic blocks once each. Function calls are skipped, and back edges are never taken so these values are often unreliable, This function is good at finding simple constant addresses which the function will use or calculate.

**Returns**
a set of constants

**property num_arguments**

**property endpoints**

**property endpoints_with_type**

**property ret_sites**

**property jumpout_sites**

**property retout_sites**

**property callout_sites**

**property size**

**property binary**
Get the object this function belongs to. :return: The object this function belongs to.

**property offset: int**

the function's binary offset (i.e., non-rebased address)

**Type**
return

**property symbol: Symbol | None**

the function's Symbol, if any

**Type**
return

**property pseudocode: str | None**

the function's pseudocode

**Type**
return

**property dirty: bool**

**mark_dirty**()

**Return type**
None

**add_jumpout_site**(*node*)

Add a custom jumpout site.

**Parameters**
**node** (*CodeNode*) – The address of the basic block that control flow leaves during this transition.

**Returns**
None

**add_retout_site**(*node*)

Add a custom retout site.

Retout (returning to outside of the function) sites are very rare. It mostly occurs during CFG recovery when we incorrectly identify the beginning of a function in the first iteration, and then correctly identify that function later in the same iteration (function alignments can lead to this bizarre case). We will mark all edges going out of the header of that function as a outside edge, because all successors now belong to the incorrectly-identified function. This identification error will be fixed in the second iteration of CFG recovery. However, we still want to keep track of jumpouts/retouts during the first iteration so other logic in CFG recovery still work.

**Parameters**
**node** (*CodeNode*) – The address of the basic block that control flow leaves the current function after a call.

**Returns**
None

**update_func_block_count**()

Update the cached block count of this function in the function manager.

**Return type**
None

`mark_nonreturning_calls_endpoints()`

Iterate through all call edges in transition graph. For each call a non-returning function, mark the source basic block as an endpoint.

This method should only be executed once all functions are recovered and analyzed by CFG recovery, so we know whether each function returns or not.

**Returns**
None

`get_call_sites()`

Gets a list of all the basic blocks that end in calls.

**Return type**
`Iterable[int]`

**Returns**
A view of the addresses of the blocks that end in calls.

`get_call_target(callsite_addr)`

Get the target of a call.

**Parameters**
**callsite_addr** – The address of a basic block that ends in a call.

**Returns**
The target of said call, or None if callsite_addr is not a callsite.

`get_call_return(callsite_addr)`

Get the hypothetical return address of a call.

**Parameters**
**callsite_addr** – The address of the basic block that ends in a call.

**Returns**
The likely return target of said call, or None if callsite_addr is not a callsite.

`property graph`

Get a local transition graph. A local transition graph is a transition graph that only contains nodes that belong to the current function. All edges, except for the edges going out from the current function or coming from outside the current function, are included.

The generated graph is cached in self._local_transition_graph.

**Returns**
A local transition graph.

**Return type**
networkx.DiGraph

`graph_ex(exception_edges=True)`

Get a local transition graph with a custom configuration. A local transition graph is a transition graph that only contains nodes that belong to the current function. This method allows user to exclude certain types of edges together with the nodes that are only reachable through such edges, such as exception edges.

The generated graph is not cached.

**Parameters**
**exception_edges** (*bool*) – Should exception edges and the nodes that are only reachable through exception edges be kept.

**Returns**
A local transition graph with a special configuration.

**Return type**
networkx.DiGraph

`transition_graph_ex`(*exception_edges=True*)

Get a transition graph with a custom configuration. This method allows user to exclude certain types of edges together with the nodes that are only reachable through such edges, such as exception edges.

The generated graph is not cached.

**Parameters**
**exception_edges** (*bool*) – Should exception edges and the nodes that are only reachable through exception edges be kept.

**Returns**
A local transition graph with a special configuration.

**Return type**
networkx.DiGraph

`subgraph`(*ins_addrs*)

Generate a sub control flow graph of instruction addresses based on self.graph

**Parameters**
**ins_addrs** (*iterable*) – A collection of instruction addresses that should be included in the subgraph.

**Return networkx.DiGraph**
A subgraph.

`instruction_size`(*insn_addr*)

Get the size of the instruction specified by *insn_addr*.

**Parameters**
**insn_addr** (*int*) – Address of the instruction

**Return int**
Size of the instruction in bytes, or None if the instruction is not found.

`addr_to_instruction_addr`(*addr*)

Obtain the address of the instruction that covers @addr.

**Parameters**
**addr** (*int*) – An address.

**Returns**
Address of the instruction that covers @addr, or None if this addr is not covered by any instruction of this function.

**Return type**
int or None

`dbg_print`()

Returns a representation of the list of basic blocks in this function.

`dbg_draw`(*filename*)

Draw the graph and save it to a PNG file.

`property arguments`

**property has_return**

**property callable**

**normalize**()

Make sure all basic blocks in the transition graph of this function do not overlap. You will end up with a CFG that IDA Pro generates.

This method does not touch the CFG result. You may call CFG{Emulated, Fast}.normalize() for that matter.

**Returns**
None

**find_declaration**(*ignore_binary_name=False*, *binary_name_hint=None*)

Find the most likely function declaration from the embedded collection of prototypes, set it to self.prototype, and update self.calling_convention with the declaration.

**Parameters**

- **ignore_binary_name** (bool) – Do not rely on the executable or library where the function belongs to determine its source library. This is useful when working on statically linked binaries (because all functions will belong to the main executable). We will search for all libraries in angr to find the first declaration match.
- **binary_name_hint** (str | None) – Substring of the library name where this function might be originally coming from. Useful for FLIRT-identified functions in statically linked binaries.

**Return type**
bool

**Returns**
True if a declaration is found and self.prototype and self.calling_convention are updated. False if we fail to find a matching function declaration, in which case self.prototype or self.calling_convention will be kept untouched.

**is_rust_function**()

Determines if the function name follows Rust mangling conventions.

**property demangled_name**

**property short_name**

**get_unambiguous_name**(*display_name=None*)

Get a disambiguated function name.

**Parameters**
**display_name** (str | None) – Name to display, otherwise the function name.

**Return type**
str

**Returns**

The function name in one of the following forms:

- ::<name> when the function binary is the main object.
- ::<obj>::<name> when the function binary is not the main object.
- ::<addr>::<name> when the function binary is an unnamed non-main object, or when multiple functions with the same name are defined in the function binary.

**apply_definition**(*definition*, *calling_convention=None*)

**Return type**
None

**Parameters**

- **definition** (*str*)
- **calling_convention** (SimCC | type[SimCC] | *None*)

**functions_reachable**()

**Return type**
set[*Function*]

**Returns**
The set of all functions that can be reached from the function represented by self.

**holes**(*min_size=8*)

Find the number of non-consecutive areas in the function that are at least *min_size* bytes large.

**Return type**
int

**Parameters**
**min_size** (*int*)

**copy**()

**pp**(***kwargs*)

Pretty-print the function disassembly.

**class** angr.knowledge_plugins.functions.**FunctionManager**

Bases: *KnowledgeBasePlugin*, Mapping[K, *Function*], Generic

When cache_limit is set, the FunctionManager uses a SpillingFunctionDict that implements an LRU cache keeping only the most recently accessed N functions in memory, spilling others to an LMDB database on disk. This allows working with binaries that have more functions than can fit in memory.

**Parameters**
**cache_limit** (int | None) – Maximum number of functions to keep in memory. None means unlimited (no eviction). Default is None.

**__init__**(*kb*, *cache_limit=None*)

**Parameters**

- **kb** (KnowledgeBase)
- **cache_limit** (*int* | *None*)

**copy**()

**clear**()

**get_default_cache_limit**(*max_limit=5000*)

Get the default function cache limit based on the size of the binary.

**Return type**
int | None

**Returns**
The default cache limit; None means unlimited.

**Parameters**
**max_limit** (*int*)

**is_plt_cached**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

**get_binary_name_cached**(*addr*)

**Return type**
str | None

**Parameters**
**addr** (*int*)

**function_name_changed**(*addr*, *old_name*, *new_name*)

Notify the FunctionManager that a function's name has changed.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **old_name** (str | None) – Old name of the function, or None if there is no old name.
- **new_name** (str) – New name of the function.

**Return type**
None

**get_by_addr**(*addr*, *meta_only=False*)

**Return type**
*Function*

**Parameters**
**meta_only** (*bool*)

**get_by_name**(*name*, *check_previous_names=False*)

**Return type**
Generator[*Function*]

**Parameters**

- **name** (*str*)
- **check_previous_names** (*bool*)

**get_addrs_by_name**(*name*, *check_previous_names=False*)

**Return type**
set[int]

**Parameters**

- **name** (*str*)
- **check_previous_names** (*bool*)

`contains_addr`(*addr*)

Decide if an address is handled by the function manager.

Note: this function is non-conformant with python programming idioms, but its needed for performance reasons.

**Parameters**
**addr** (*int*) – Address of the function.

`ceiling_addr`(*addr*)

Return the function who has the least address that is greater than or equal to *addr*.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – The address to query.

**Return type**
Optional[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
A Function instance, or None if there is no other function after *addr*.

`ceiling_func`(*addr*)

Return the function who has the least address that is greater than or equal to *addr*.

**Parameters**
**addr** (*int*) – The address to query.

**Return type**
*Function* | None

**Returns**
A Function instance, or None if there is no other function after *addr*.

`floor_addr`(*addr*)

Return the function who has the greatest address that is less than or equal to *addr*.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – The address to query.

**Return type**
Optional[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
An address, or None if there is no other function before *addr*.

`floor_func`(*addr*)

Return the function who has the greatest address that is less than or equal to *addr*.

**Parameters**
**addr** (*int*) – The address to query.

**Returns**
A Function instance, or None if there is no other function before *addr*.

**Return type**
*Function* or None

`query`(*query*, *check_previous_names=False*)

Query for a function using selectors to disambiguate. Supported variations:

::<name> Function <name> in the main object ::<addr>::<name> Function <name> at <addr> ::<obj>::<name> Function <name> in <obj>

**Return type**
*Function* | None

**Parameters**

- **query** (*str*)
- **check_previous_names** (*bool*)

**function**(*addr=None*, *name=None*, *check_previous_names=False*, *create=False*, *syscall=False*, *plt=None*)

Get a function object from the function manager.

Pass either *addr* or *name* with the appropriate values.

**Parameters**

- **addr** (Optional[TypeVar(K, int, SootMethodDescriptor)]) – Address of the function.
- **name** (str | None) – Name of the function.
- **create** (bool) – Whether to create the function or not if the function does not exist.
- **syscall** (bool) – True to create the function as a syscall, False otherwise.
- **plt** (bool | None) – True to find the PLT stub, False to find a non-PLT stub, None to disable this restriction.
- **check_previous_names** (*bool*)

**Returns**
The Function instance, or None if the function is not found and create is False.

**Return type**
*Function* | None

**dbg_draw**(*prefix='dbg_function_'*)

**rebuild_callgraph**()

**set_function_returning**(*addr*, *v*)

**Return type**
None

**Parameters**

- **addr** (*K*)
- **v** (*bool | None*)

**nonreturning_func_addrs**()

Yield all non-returning function addresses.

**Return type**
Generator[int]

**unknown_returning_func_addrs**()

Yield all function addresses with unknown returning status.

**Return type**
Generator[int]

`is_func_nonreturning`(*addr*)

Check if a function is non-returning.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
bool

**Returns**
True if non-returning, False if returning or unknown.

`is_func_returning_unknown`(*addr*)

Check if a function's returning status is unknown.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
bool

**Returns**
True if returning status is unknown, False otherwise.

`get_func_block_count`(*addr*)

Get the number of blocks in a function.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
int | None

**Returns**
Number of blocks, or None if unknown.

`set_func_block_count`(*addr*, *count*)

Set the number of blocks in a function.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **count** (int) – Number of blocks.

**Return type**
None

**Returns**
None

`get_func_addr_with_block_count`(*block_count*)

Get the addresses of all functions that have exactly `block_count` blocks.

**Parameters**
**block_count** (int) – The number of blocks to match.

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
The set of matching function addresses.

`get_func_name`(*addr*)

Get the name of a function without loading the (possibly spilled) Function object.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
str | None

**Returns**
The function's name, or None if unknown.

`get_from_signature`(*addr*)

Get the `from_signature` value of a function without loading the (possibly spilled) Function object.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
str | None

**Returns**
The function's `from_signature` value (e.g. `"flirt"`), or None.

`set_from_signature`(*addr*, *from_signature*)

Update the cached `from_signature` value for a function. Called by Function.from_signature's setter.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **from_signature** (str | None) – The new `from_signature` value.

**Return type**
None

**Returns**
None

`get_all_funcaddrs_from_signature`(*from_signature='flirt'*)

Get the addresses of all functions whose `from_signature` matches, without loading any Function object.

**Parameters**
**from_signature** (str) – The `from_signature` value to match (default `"flirt"`).

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
The set of matching function addresses.

`get_key_func_addrs`(*func_type*)

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Parameters**
**func_type** (str)

`get_key_func_type_and_addrs`()

Get all cached key functions grouped by key-function type (e.g. `"alloca_probe"`, `"jmp_rax"`).

**Return type**
dict[str, set[TypeVar(K, int, SootMethodDescriptor)]]

**Returns**
A mapping from key-function type to the set of function addresses of that type.

**add_key_func_addr**(*func_type*, *addr*)

**Return type**
None

**Parameters**

- **func_type** (*str*)
- **addr** (*K*)

**index_key_func_addrs**(*func*)

Record every key-function flag that `func.info` carries in the key-function address cache.

**Return type**
None

**Parameters**
**func** (Function)

**property cache_limit: int | None**

Get the maximum number of functions to keep in memory. None means unlimited (no eviction).

**property cached_function_count: int**

Return the number of functions currently in memory.

**property spilled_function_count: int**

Return the number of functions currently spilled to LMDB.

**property total_function_count: int**

Return the total number of functions (in memory + spilled).

**class** angr.knowledge_plugins.functions.**PrototypeSource**

Bases: int, Enum

An enumeration to represent the source of a function prototype. The values are ordered by certainty, with higher values indicating more certainty.

**NONE = 0**

**GUESSED = 1**

**CCA_LOW = 2**

**CALLSITE_LOW = 3**

**CALLSITE_DECOMPILER = 6**

**CCA_DECOMPILER = 20**

**SIMPROC = 21**

**SIGNATURES = 25**

**USER = 100**

**__new__**(*value*)

### Submodules


*function*
*function_manager*
*function_parser*
*soot_function*


### angr.knowledge_plugins.functions.function

angr.knowledge_plugins.functions.function.**dirty_func**(*func*)

**class** angr.knowledge_plugins.functions.function.**PrototypeSource**

Bases: int, Enum

An enumeration to represent the source of a function prototype. The values are ordered by certainty, with higher values indicating more certainty.

**NONE = 0**

**GUESSED = 1**

**CCA_LOW = 2**

**CALLSITE_LOW = 3**

**CALLSITE_DECOMPILER = 6**

**CCA_DECOMPILER = 20**

**SIMPROC = 21**

**SIGNATURES = 25**

**USER = 100**

**__new__**(*value*)

**class** angr.knowledge_plugins.functions.function.**FunctionInfo**

Bases: UserDict

A dictionary that updates the .dirty field of the Function object when modified.

**__init__**(*func*)

**Parameters**
**func** (Function)

**copy**(*owner*)

**Return type**
*FunctionInfo*

**Parameters**
**owner** (Function)

**to_json**()

**Return type**
str

**classmethod from_json**(*json_str*, *owner*)

**Return type**
*FunctionInfo*

**Parameters**

- **json_str** (*str*)
- **owner** (Function)

**class** angr.knowledge_plugins.functions.function.**Function**

Bases: *Serializable*

A representation of a function and various information about it.

**Variables**

- **meta_only** (bool) – Whether this function only contains meta-information and in read-only mode.
- **_dirty** (bool) – Whether this function has been modified since last serialization.
- **evicted** (bool) – Whether this function has been evicted from FunctionManager to external storage.

**__init__**(*function_manager*, *addr*, *name=None*, *syscall=None*, *is_simprocedure=None*, *binary_name=None*, *is_plt=None*, *returning=None*, *alignment=False*, *calling_convention=None*, *prototype=None*, *prototype_libname=None*, *prototype_source=None*, *is_prototype_guessed=True*)

Function constructor. If the optional parameters are not provided, they will be automatically determined upon the creation of a Function object.

**Parameters**

- **addr** (int) – The address of the function.
- **name** (*str*) – The name of the function.
- **syscall** (*bool*) – Whether this function is a syscall or not.
- **is_simprocedure** (bool | None) – Whether this function is a SimProcedure or not.
- **binary_name** (*str*) – Name of the binary where this function is.
- **is_plt** (bool | None) – If this function is a PLT entry.
- **returning** (*bool*) – If this function returns.
- **alignment** (*bool*) – If this function acts as an alignment filler. Such functions usually only contain nops.
- **function_manager** (FunctionManager | *None*)
- **calling_convention** (SimCC | *None*)
- **prototype** (SimTypeFunction | *None*)
- **prototype_libname** (*str* | *None*)
- **prototype_source** (PrototypeSource | *None*)
- **is_prototype_guessed** (*bool*)

**transition_graph**

**normalized**

**addr**

**startpoint**

**bp_on_stack**

**retaddr_on_stack**

**sp_delta**

**tags**

**ran_cca**

**meta_only:** *bool*

**evicted:** *bool*

**is_default_name**

**previous_names**

**binary_name**

**property name**

**property from_signature:** *str* | *None*

**property project:** *Project* | *None*

**property returning**

**property calling_convention:** *SimCC* | *None*

**property prototype:** *SimTypeFunction* | *None*

**property prototype_libname**

**property is_prototype_guessed:** *bool*

**property is_prototype_groundtruth:** *bool*

True if the prototype comes from outside of the decompiler (SimProcedures, signatures, or the user) and may be fed back into type inference as ground truth. Prototypes inferred by the decompiler itself are excluded so that re-decompiling a function does not freeze its own earlier guess.

**property prototype_source:** *PrototypeSource*

**property info:** *FunctionInfo*

**property is_plt:** *bool*

**property is_simprocedure:** *bool*

**property is_syscall:** *bool*

**property is_alignment:** *bool*

**property blocks**

An iterator of all local blocks in the current function.

**Returns**
angr.lifter.Block instances.

property code_nodes: dict[int, CodeNode]

property cyclomatic_complexity

The cyclomatic complexity of the function.

Cyclomatic complexity is a software metric used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code. It is computed using the formula: M = E - N + 2P, where E = the number of edges in the graph, N = the number of nodes in the graph, P = the number of connected components.

The cyclomatic complexity value is lazily computed and cached for future use. Initially this value is None until it is computed for the first time

**Returns**
The cyclomatic complexity of the function.

**Return type**
int

property xrefs: Iterator[XRef]

An iterator of all xrefs of the current function.

**Returns**
angr.knowledge_plugins.xrefs.xref.XRef instances.

property block_addrs

An iterator of all local block addresses in the current function.

**Returns**
block addresses.

property block_addrs_set

Return a set of block addresses for a better performance of inclusion tests.

**Returns**
A set of block addresses.

**Return type**
set

get_block(*addr*, *size=None*, *byte_string=None*)

Getting a block out of the current function.

**Parameters**

- **addr** (int) – The address of the block.
- **size** (int | None) – The size of the block. This is optional. If not provided, angr will load
- **byte_string** (bytes | None)

**Returns**

get_block_size(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*int*)

property nodes: Iterable[CodeNode]

**get_node**(*addr*)

**Return type**
*BlockNode* | None

**property has_unresolved_jumps**

**property has_unresolved_calls**

**property operations**
All of the operations that are done by this functions.

**property code_constants**
All of the constants that are used by this functions's code.

**classmethod parse_from_cmessage**(*cmsg*, ***kwargs*)

**Parameters**
**cmsg**

**Return Function**
The function instantiated out of the cmsg data.

**string_references**(*minimum_length=2*)
All of the constant string references used by this function.

**Parameters**
**minimum_length** – The minimum length of strings to find (default is 1)

**Returns**
A generator yielding tuples of (address, string) where is address is the location of the string in memory.

**property local_runtime_values**
Tries to find all runtime values of this function which do not come from inputs. These values are generated by starting from a blank state and reanalyzing the basic blocks once each. Function calls are skipped, and back edges are never taken so these values are often unreliable, This function is good at finding simple constant addresses which the function will use or calculate.

**Returns**
a set of constants

**property num_arguments**

**property endpoints**

**property endpoints_with_type**

**property ret_sites**

**property jumpout_sites**

**property retout_sites**

**property callout_sites**

**property size**

**property binary**
Get the object this function belongs to. :return: The object this function belongs to.

property offset: int

the function's binary offset (i.e., non-rebased address)

**Type**
return

property symbol: Symbol | None

the function's Symbol, if any

**Type**
return

property pseudocode: str | None

the function's pseudocode

**Type**
return

property dirty: bool

mark_dirty()

**Return type**
None

add_jumpout_site(*node*)

Add a custom jumpout site.

**Parameters**
**node** (*CodeNode*) – The address of the basic block that control flow leaves during this transition.

**Returns**
None

add_retout_site(*node*)

Add a custom retout site.

Retout (returning to outside of the function) sites are very rare. It mostly occurs during CFG recovery when we incorrectly identify the beginning of a function in the first iteration, and then correctly identify that function later in the same iteration (function alignments can lead to this bizarre case). We will mark all edges going out of the header of that function as a outside edge, because all successors now belong to the incorrectly-identified function. This identification error will be fixed in the second iteration of CFG recovery. However, we still want to keep track of jumpouts/retouts during the first iteration so other logic in CFG recovery still work.

**Parameters**
**node** (*CodeNode*) – The address of the basic block that control flow leaves the current function after a call.

**Returns**
None

update_func_block_count()

Update the cached block count of this function in the function manager.

**Return type**
None

**mark_nonreturning_calls_endpoints**()

Iterate through all call edges in transition graph. For each call a non-returning function, mark the source basic block as an endpoint.

This method should only be executed once all functions are recovered and analyzed by CFG recovery, so we know whether each function returns or not.

**Returns**
None

**get_call_sites**()

Gets a list of all the basic blocks that end in calls.

**Return type**
Iterable[int]

**Returns**
A view of the addresses of the blocks that end in calls.

**get_call_target**(*callsite_addr*)

Get the target of a call.

**Parameters**
**callsite_addr** – The address of a basic block that ends in a call.

**Returns**
The target of said call, or None if callsite_addr is not a callsite.

**get_call_return**(*callsite_addr*)

Get the hypothetical return address of a call.

**Parameters**
**callsite_addr** – The address of the basic block that ends in a call.

**Returns**
The likely return target of said call, or None if callsite_addr is not a callsite.

**property graph**

Get a local transition graph. A local transition graph is a transition graph that only contains nodes that belong to the current function. All edges, except for the edges going out from the current function or coming from outside the current function, are included.

The generated graph is cached in self._local_transition_graph.

**Returns**
A local transition graph.

**Return type**
networkx.DiGraph

**graph_ex**(*exception_edges=True*)

Get a local transition graph with a custom configuration. A local transition graph is a transition graph that only contains nodes that belong to the current function. This method allows user to exclude certain types of edges together with the nodes that are only reachable through such edges, such as exception edges.

The generated graph is not cached.

**Parameters**
**exception_edges** (*bool*) – Should exception edges and the nodes that are only reachable through exception edges be kept.

**Returns**
A local transition graph with a special configuration.

**Return type**
networkx.DiGraph

`transition_graph_ex`(*exception_edges=True*)

Get a transition graph with a custom configuration. This method allows user to exclude certain types of edges together with the nodes that are only reachable through such edges, such as exception edges.

The generated graph is not cached.

**Parameters**
**exception_edges** (*bool*) – Should exception edges and the nodes that are only reachable through exception edges be kept.

**Returns**
A local transition graph with a special configuration.

**Return type**
networkx.DiGraph

`subgraph`(*ins_addrs*)

Generate a sub control flow graph of instruction addresses based on self.graph

**Parameters**
**ins_addrs** (*iterable*) – A collection of instruction addresses that should be included in the subgraph.

**Return networkx.DiGraph**
A subgraph.

`instruction_size`(*insn_addr*)

Get the size of the instruction specified by *insn_addr*.

**Parameters**
**insn_addr** (*int*) – Address of the instruction

**Return int**
Size of the instruction in bytes, or None if the instruction is not found.

`addr_to_instruction_addr`(*addr*)

Obtain the address of the instruction that covers @addr.

**Parameters**
**addr** (*int*) – An address.

**Returns**
Address of the instruction that covers @addr, or None if this addr is not covered by any instruction of this function.

**Return type**
int or None

`dbg_print`()

Returns a representation of the list of basic blocks in this function.

`dbg_draw`(*filename*)

Draw the graph and save it to a PNG file.

`property arguments`

**property has_return**

**property callable**

**normalize**()

Make sure all basic blocks in the transition graph of this function do not overlap. You will end up with a CFG that IDA Pro generates.

This method does not touch the CFG result. You may call CFG{Emulated, Fast}.normalize() for that matter.

**Returns**
None

**find_declaration**(*ignore_binary_name=False*, *binary_name_hint=None*)

Find the most likely function declaration from the embedded collection of prototypes, set it to self.prototype, and update self.calling_convention with the declaration.

**Parameters**

- **ignore_binary_name** (bool) – Do not rely on the executable or library where the function belongs to determine its source library. This is useful when working on statically linked binaries (because all functions will belong to the main executable). We will search for all libraries in angr to find the first declaration match.
- **binary_name_hint** (str | None) – Substring of the library name where this function might be originally coming from. Useful for FLIRT-identified functions in statically linked binaries.

**Return type**
bool

**Returns**
True if a declaration is found and self.prototype and self.calling_convention are updated. False if we fail to find a matching function declaration, in which case self.prototype or self.calling_convention will be kept untouched.

**is_rust_function**()

Determines if the function name follows Rust mangling conventions.

**property demangled_name**

**property short_name**

**get_unambiguous_name**(*display_name=None*)

Get a disambiguated function name.

**Parameters**
**display_name** (str | None) – Name to display, otherwise the function name.

**Return type**
str

**Returns**

The function name in one of the following forms:

- ::<name> when the function binary is the main object.
- ::<obj>::<name> when the function binary is not the main object.
- ::<addr>::<name> when the function binary is an unnamed non-main object, or when multiple functions with the same name are defined in the function binary.

**apply_definition**(*definition*, *calling_convention=None*)

**Return type**
None

**Parameters**

- **definition** (*str*)
- **calling_convention** (SimCC | type[SimCC] | *None*)

**functions_reachable**()

**Return type**
set[*Function*]

**Returns**
The set of all functions that can be reached from the function represented by self.

**holes**(*min_size=8*)

Find the number of non-consecutive areas in the function that are at least *min_size* bytes large.

**Return type**
int

**Parameters**
**min_size** (*int*)

**copy**()

**pp**(***kwargs*)

Pretty-print the function disassembly.

## angr.knowledge_plugins.functions.function_manager

**class** angr.knowledge_plugins.functions.function_manager.**FunctionDictBase**

Bases: Generic

Base class for FunctionDict and SpillingFunctionDict.

**__init__**(*backref*, *key_types=<class 'int'>*)

**Parameters**

- **backref** (FunctionManager[K] | *None*)
- **key_types** (type)

**floor_addr**(*addr*)

**Parameters**
**addr** (*K*)

**ceiling_addr**(*addr*)

**items**()

**get**(*addr*, *default=<object object>*, */*, *meta_only=False*)

**Overloads**

- **self**, **key** (K), **default** (None), **meta_only** (bool) → Function
- **self**, **key** (K), **default** (Function), **meta_only** (bool) → Function

- **self**, **key** (K), **default** (T), **meta_only** (bool) → Function | T

**Parameters**

- **addr** (*K*)
- **meta_only** (*bool*)

**irange**(*minimum=None*, *maximum=None*, *inclusive=(True, True)*, *reverse=False*)

**class** angr.knowledge_plugins.functions.function_manager.**FunctionDict**

Bases: SortedDict[K, *Function*], *FunctionDictBase*[K]

FunctionDict is a dict where the keys are function starting addresses and map to the associated Function.

**floor_addr**(*addr*)

**Parameters**
**addr** (*K*)

**ceiling_addr**(*addr*)

**class** angr.knowledge_plugins.functions.function_manager.**SpillingFunctionDict**

Bases: UserDict[K, *Function*], *FunctionDictBase*[K]

SpillingFunctionDict extends FunctionDict with LRU caching and LMDB spilling. This class keeps only the most recently accessed N functions in memory, spilling others to an LMDB database on disk.

SpillingFunctionDict also keeps a cache of meta-only Function objects that do not have graph or block information. These meta-only Function objects are read-only and may become stale if the full Function is later loaded and updated. Therefore, please be extremely cautious when using these meta-only Function objects.

SpillingFunctionDict._load_from_lmdb() does not support reentry. If a function being loaded from LMDB triggers another load from LMDB, the inner load will raise a RuntimeError.

A Function instance becomes "dirty" (Function.dirty) if it has been modified since being loaded from LMDB. Dirty functions are always saved back to LMDB when evicted, while clean functions are skipped during eviction.

A Function instance becomes "evicted" (Function.evicted) and stale if it has been spilled to LMDB. An evicted Function instance may not reflect the most recent state of the function, and any changes to an evicted Function instance will not be saved to LMDB. Therefore, if SpillingFunctionDict is in use, it is advised not to hold a Function instance for too long before using it.

**Variables**

- **cache_limit** – The maximum number of functions to keep in memory.
- **rtdb** – A reference to the RuntimeDb knowledge base plugin.
- **_lru_order** – An OrderedDict tracking the eviction order of cached functions.
- **_spilled_keys** – A set of function addresses that have been spilled to LMDB.
- **_db_batch_size** – The number of functions that are evicted in a single batch.
- **_meta_func_cache** – An LRU cache for meta-only Function objects.
- **_eviction_enabled** – A flag indicating whether eviction is currently enabled or not.

**__init__**(*backref*, *rtdb*, */*, *key_types=<class 'int'>*, *cache_limit=1000*, *db_batch_size=100*, ***kwargs*)

**Parameters**

- **backref** (FunctionManager*[K] | None*)
- **rtdb** (RuntimeDb)

- **key_types** (type)
- **cache_limit** (*int*)
- **db_batch_size** (*int*)

**copy**()

**Return type**
*SpillingFunctionDict*[TypeVar(K, int, SootMethodDescriptor)]

**clear**()

Clear all functions from memory and spilled storage.

**Return type**
None

**property cached_keys:** Generator[K]

**property cache_limit:** int

Get the maximum number of functions to keep in memory.

**property cached_count:** int

Return the number of functions currently in memory.

**property spilled_count:** int

Return the number of functions currently spilled to LMDB.

**property total_count:** int

Return the total number of functions (in memory + spilled).

**is_cached**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*K*)

**bulk_import_serialized**(*items*)

Bulk-import already-serialized functions directly into the LMDB backing store and register them as spilled, without deserializing them. The serialized bytes must be serialized function_pb2.Function messages, i.e., the exact format that _save_to_lmdb() writes.

Imported functions are deserialized lazily upon first access, through the regular _load_from_lmdb() path.

**Parameters**
**items** (list[tuple[TypeVar(K, int, SootMethodDescriptor), bytes]]) – A list of (address, serialized function bytes) tuples.

**Return type**
None

**export_serialized**()

Export all functions as serialized bytes, e.g., for dumping into an angr database.

Functions whose LMDB records are guaranteed to be current are copied directly from the LMDB backing store (in a single read transaction) without being deserialized and re-serialized. This applies to all spilled functions and to cached functions that are clean: a Function instance is only ever clean if it was deserialized from serialized bytes and has not been modified since (eviction relies on the same invariant to skip writing clean functions back to LMDB). Dirty functions, and functions whose LMDB record is missing, are serialized from their in-memory objects through the regular path.

**Return type**
list[tuple[TypeVar(K, int, SootMethodDescriptor), bytes, bool]]

**Returns**
A list of (address, serialized function bytes, copied_from_lmdb) tuples, ordered by address.

**load_all_spilled**()

Load all spilled functions back into memory (disables eviction temporarily).

**Return type**
None

**evict_all_cached**()

Evict all cached functions to LMDB.

**Return type**
None

**class** angr.knowledge_plugins.functions.function_manager.**FunctionManager**

Bases: *KnowledgeBasePlugin*, Mapping[K, *Function*], Generic

When cache_limit is set, the FunctionManager uses a SpillingFunctionDict that implements an LRU cache keeping only the most recently accessed N functions in memory, spilling others to an LMDB database on disk. This allows working with binaries that have more functions than can fit in memory.

**Parameters**
**cache_limit** (int | None) – Maximum number of functions to keep in memory. None means unlimited (no eviction). Default is None.

**__init__**(*kb*, *cache_limit=None*)

**Parameters**

- **kb** (KnowledgeBase)
- **cache_limit** (*int | None*)

**copy**()

**clear**()

**get_default_cache_limit**(*max_limit=5000*)

Get the default function cache limit based on the size of the binary.

**Return type**
int | None

**Returns**
The default cache limit; None means unlimited.

**Parameters**
**max_limit** (*int*)

**is_plt_cached**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*int*)

get_binary_name_cached(*addr*)

**Return type**
str | None

**Parameters**
**addr** (*int*)

function_name_changed(*addr*, *old_name*, *new_name*)

Notify the FunctionManager that a function's name has changed.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **old_name** (str | None) – Old name of the function, or None if there is no old name.
- **new_name** (str) – New name of the function.

**Return type**
None

get_by_addr(*addr*, *meta_only=False*)

**Return type**
*Function*

**Parameters**
**meta_only** (*bool*)

get_by_name(*name*, *check_previous_names=False*)

**Return type**
Generator[*Function*]

**Parameters**

- **name** (*str*)
- **check_previous_names** (*bool*)

get_addrs_by_name(*name*, *check_previous_names=False*)

**Return type**
set[int]

**Parameters**

- **name** (*str*)
- **check_previous_names** (*bool*)

contains_addr(*addr*)

Decide if an address is handled by the function manager.

Note: this function is non-conformant with python programming idioms, but its needed for performance reasons.

**Parameters**
**addr** (*int*) – Address of the function.

ceiling_addr(*addr*)

Return the function who has the least address that is greater than or equal to *addr*.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – The address to query.

**Return type**
Optional[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
A Function instance, or None if there is no other function after *addr*.

**ceiling_func**(*addr*)

Return the function who has the least address that is greater than or equal to *addr*.

**Parameters**
**addr** (*int*) – The address to query.

**Return type**
*Function* | None

**Returns**
A Function instance, or None if there is no other function after *addr*.

**floor_addr**(*addr*)

Return the function who has the greatest address that is less than or equal to *addr*.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – The address to query.

**Return type**
Optional[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
An address, or None if there is no other function before *addr*.

**floor_func**(*addr*)

Return the function who has the greatest address that is less than or equal to *addr*.

**Parameters**
**addr** (*int*) – The address to query.

**Returns**
A Function instance, or None if there is no other function before *addr*.

**Return type**
*Function* or None

**query**(*query*, *check_previous_names=False*)

Query for a function using selectors to disambiguate. Supported variations:

::<name> Function <name> in the main object ::<addr>::<name> Function <name> at <addr> ::<obj>::<name> Function <name> in <obj>

**Return type**
*Function* | None

**Parameters**

- **query** (*str*)
- **check_previous_names** (*bool*)

**function**(*addr=None*, *name=None*, *check_previous_names=False*, *create=False*, *syscall=False*, *plt=None*)

Get a function object from the function manager.

Pass either *addr* or *name* with the appropriate values.

**Parameters**

- **addr** (Optional[TypeVar(K, int, SootMethodDescriptor)]) – Address of the function.
- **name** (str | None) – Name of the function.
- **create** (bool) – Whether to create the function or not if the function does not exist.
- **syscall** (bool) – True to create the function as a syscall, False otherwise.
- **plt** (bool | None) – True to find the PLT stub, False to find a non-PLT stub, None to disable this restriction.
- **check_previous_names** (*bool*)

**Returns**

The Function instance, or None if the function is not found and create is False.

**Return type**

*Function* | None

**dbg_draw**(*prefix='dbg_function_'*)

**rebuild_callgraph**()

**set_function_returning**(*addr*, *v*)

**Return type**

None

**Parameters**

- **addr** (*K*)
- **v** (*bool | None*)

**nonreturning_func_addrs**()

Yield all non-returning function addresses.

**Return type**

Generator[int]

**unknown_returning_func_addrs**()

Yield all function addresses with unknown returning status.

**Return type**

Generator[int]

**is_func_nonreturning**(*addr*)

Check if a function is non-returning.

**Parameters**

**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**

bool

**Returns**

True if non-returning, False if returning or unknown.

**is_func_returning_unknown**(*addr*)

Check if a function's returning status is unknown.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
bool

**Returns**
True if returning status is unknown, False otherwise.

**get_func_block_count**(*addr*)

Get the number of blocks in a function.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
int | None

**Returns**
Number of blocks, or None if unknown.

**set_func_block_count**(*addr*, *count*)

Set the number of blocks in a function.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **count** (int) – Number of blocks.

**Return type**
None

**Returns**
None

**get_func_addr_with_block_count**(*block_count*)

Get the addresses of all functions that have exactly `block_count` blocks.

**Parameters**
**block_count** (int) – The number of blocks to match.

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
The set of matching function addresses.

**get_func_name**(*addr*)

Get the name of a function without loading the (possibly spilled) Function object.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
str | None

**Returns**
The function's name, or None if unknown.

get_from_signature(*addr*)

Get the from_signature value of a function without loading the (possibly spilled) Function object.

**Parameters**
**addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.

**Return type**
str | None

**Returns**
The function's from_signature value (e.g. "flirt"), or None.

set_from_signature(*addr*, *from_signature*)

Update the cached from_signature value for a function. Called by Function.from_signature's setter.

**Parameters**

- **addr** (TypeVar(K, int, SootMethodDescriptor)) – Address of the function.
- **from_signature** (str | None) – The new from_signature value.

**Return type**
None

**Returns**
None

get_all_funcaddrs_from_signature(*from_signature='flirt'*)

Get the addresses of all functions whose from_signature matches, without loading any Function object.

**Parameters**
**from_signature** (str) – The from_signature value to match (default "flirt").

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Returns**
The set of matching function addresses.

get_key_func_addrs(*func_type*)

**Return type**
set[TypeVar(K, int, SootMethodDescriptor)]

**Parameters**
**func_type** (*str*)

get_key_func_type_and_addrs()

Get all cached key functions grouped by key-function type (e.g. "alloca_probe", "jmp_rax").

**Return type**
dict[str, set[TypeVar(K, int, SootMethodDescriptor)]]

**Returns**
A mapping from key-function type to the set of function addresses of that type.

add_key_func_addr(*func_type*, *addr*)

**Return type**
None

**Parameters**

- **func_type** (*str*)

- **addr** (*K*)

**index_key_func_addrs**(*func*)

Record every key-function flag that `func.info` carries in the key-function address cache.

**Return type**
None

**Parameters**
**func** (Function)

**property cache_limit: int | None**

Get the maximum number of functions to keep in memory. None means unlimited (no eviction).

**property cached_function_count: int**

Return the number of functions currently in memory.

**property spilled_function_count: int**

Return the number of functions currently spilled to LMDB.

**property total_function_count: int**

Return the total number of functions (in memory + spilled).

### angr.knowledge_plugins.functions.function_parser

**class** angr.knowledge_plugins.functions.function_parser.**CallingConventionSerializer**

Bases: object

Serialize/deserialize SimCC classes.

**static to_json**(*cc*)

**Return type**
dict

**Parameters**
**cc** (SimCC)

**static from_json**(*data*, *arch*)

**Return type**
*SimCC* | None

**Parameters**
**data** (*dict*)

**class** angr.knowledge_plugins.functions.function_parser.**FunctionParser**

Bases: object

The implementation of the serialization methods for the <Function> class.

**static serialize**(*function*)

:return :

**static parse_from_cmsg**(*cmsg*, *function_manager=None*, *project=None*, *meta_only=False*)

**Parameters**

- **cmsg** – The data to instantiate the <Function> from.
- **meta_only** (*bool*)

**Return Function**

#### angr.knowledge_plugins.functions.soot_function

**class** angr.knowledge_plugins.functions.soot_function.**SootFunction**

Bases: *Function*

A representation of a function and various information about it.

**__init__**(*function_manager*, *addr*, *name=None*, *syscall=None*)

Function constructor for Soot

**Parameters**

- **addr** – The address of the function.
- **name** – (Optional) The name of the function.
- **syscall** – (Optional) Whether this function is a syscall or not.

### 10.22.9 angr.knowledge_plugins.indirect_jumps

**class** angr.knowledge_plugins.indirect_jumps.**IndirectJumps**

Bases: *KnowledgeBasePlugin*, dict

This plugin tracks the targets of indirect jumps

**update_resolved_addrs**(*indirect_address*, *resolved_addresses*)

**Parameters**

- **indirect_address** (*int*)
- **resolved_addresses** (*list[int]*)

### 10.22.10 angr.knowledge_plugins.key_definitions

**class** angr.knowledge_plugins.key_definitions.**Definition**

Bases: Generic

An atom definition.

**Variables**

- **atom** – The atom being defined.
- **codeloc** – Where this definition is created in the original binary code.
- **dummy** – Tell whether the definition should be considered dummy or not. During simplification by AILment, definitions marked as dummy will not be removed.
- **tags** – A set of tags containing information about the definition gathered during analyses.

**__init__**(*atom*, *codeloc*, *dummy=False*, *tags=None*)

**Parameters**

- **atom** (*A*)
- **codeloc** (*CodeLoc*)
- **dummy** (*bool*)
- **tags** (*set[Tag] | None*)

**atom**

**codeloc**

**dummy**

**tags**

**property offset:** *int*

**property size:** *int*

**matches**(***kwargs*)

Return whether this definition has certain characteristics.

**Return type**
*bool*

**class** angr.knowledge_plugins.key_definitions.**DerefSize**

Bases: Enum

An enum for specialized kinds of dereferences

**NULL_TERMINATE - Dereference until the first byte which could be a literal null. Return a value including the**
terminator.

**NULL_TERMINATE = 1**

**class** angr.knowledge_plugins.key_definitions.**KeyDefinitionManager**

Bases: *KnowledgeBasePlugin*

KeyDefinitionManager manages and caches reaching definition models for each function.

For each function, by default we cache the entire reaching definitions model with observed results at the following locations: - Before each call instruction: ('insn', address of the call instruction, OP_BEFORE) - After returning from each call: ('node', address of the block that ends with a call, OP_AFTER)

**__init__**(*kb*)

**Parameters**
**kb** (*KnowledgeBase*)

**has_model**(*func_addr*)

**Parameters**
**func_addr** (*int*)

**get_model**(*func_addr*)

**Parameters**
**func_addr** (*int*)

**copy**()

**Return type**
*KeyDefinitionManager*

**class** angr.knowledge_plugins.key_definitions.**LiveDefinitions**

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

INITIAL_SP_16BIT = 32512

INITIAL_SP_32BIT = 2147418112

INITIAL_SP_64BIT = 140737488289792

__init__(*arch*, *track_tmps=False*, *canonical_size=8*, *registers=None*, *stack=None*, *memory=None*, *heap=None*, *tmps=None*, *others=None*, *register_uses=None*, *stack_uses=None*, *heap_uses=None*, *memory_uses=None*, *tmp_uses=None*, *other_uses=None*, *element_limit=5*, *merge_into_tops=True*)

**Parameters**

- **arch** (*Arch*)
- **track_tmps** (*bool*)
- **merge_into_tops** (*bool*)

project: *Project* | None

arch

track_tmps

registers: *MultiValuedMemory*

stack: *MultiValuedMemory*

memory: *MultiValuedMemory*

heap: *MultiValuedMemory*

tmps: dict[int, set[*Definition*]]

others: dict[*Atom*, *MultiValues*]

register_uses

stack_uses

heap_uses

memory_uses

tmp_uses: dict[int, set[*CodeLocation*]]

other_uses

uses_by_codeloc: dict[*CodeLocation*, set[*Definition*]]

copy(*discard_tmpdefs=False*)

**Return type**
*LiveDefinitions*

reset_uses()

static top(*bits*)

Get a TOP value.

**Parameters**
**bits** (int) – Width of the TOP value (in bits).

**Returns**
The TOP value.

static is_top(*expr*)

Check if the given expression is a TOP value.

**Parameters**
**expr** – The given expression.

**Return type**
bool

**Returns**
True if the expression is TOP, False otherwise.

stack_address(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int*)

static is_stack_address(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

static get_stack_offset(*addr*, *had_stack_base=False*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

static annotate_with_def(*symvar*, *definition*)

**Parameters**

- **symvar** (TypeVar(MVType, bound= BV | FP))
- **definition** (*Definition*)

**Return type**
TypeVar(MVType, bound= BV | FP)

**Returns**

static extract_defs(*symvar*)

**Return type**
Generator[*Definition*]

**Parameters**
**symvar** (*Base*)

**static extract_defs_from_annotations**(*annos*)

**Return type**
set[*Definition*]

**Parameters**
**annos** (*Iterable[Annotation]*)

**static extract_defs_from_mv**(*mv*)

**Return type**
Generator[*Definition*]

**Parameters**
**mv** (MultiValues)

**get_sp**()

Return the concrete value contained by the stack pointer.

**Return type**
int

**get_sp_offset**()

Return the offset of the stack pointer.

**Return type**
int | None

**get_stack_address**(*offset*)

**Return type**
int | None

**Parameters**
**offset** (*Base*)

**stack_offset_to_stack_addr**(*offset*)

**Return type**
int

**merge**(**others*)

**Return type**
tuple[*LiveDefinitions*, bool]

**Parameters**
**others** (LiveDefinitions)

**compare**(*other*)

**Return type**
bool

**Parameters**
**other** (LiveDefinitions)

**kill_definitions**(*atom*)

Overwrite existing definitions w.r.t 'atom' with a dummy definition instance. A dummy definition will not be removed during simplification.

**Parameters**
**atom** (*Atom*)

**Return type**
None

**Returns**
None

**kill_and_add_definition**(*atom*, *code_loc*, *data*, *dummy=False*, *tags=None*, *endness=None*, *annotated=False*)

**Return type**
*MultiValues* | None

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **data** (MultiValues)
- **tags** (*set[Tag] | None*)

**add_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_use_by_def**(*definition*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **definition** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any*)

**get_definitions**(*thing*)

**get_tmp_definitions**(*tmp_idx*)

**Return type**
set[*Definition*]

**Parameters**
**tmp_idx** (*int*)

`get_register_definitions`(*reg_offset*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)

`get_stack_values`(*stack_offset*, *size*, *endness*)

**Return type**
*MultiValues* | None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **endness** (*str*)

`get_stack_definitions`(*stack_offset*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)

`get_heap_definitions`(*heap_addr*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **heap_addr** (*int*)
- **size** (*int*)

`get_memory_definitions`(*addr*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **addr** (*int*)
- **size** (*int*)

`get_values`(*spec*, *endness=None*)

`get_one_value`(*spec*, *strip_annotations=False*)

`get_concrete_value`(*spec*, *cast_to=<class 'int'>*)

**Overloads**

- **self**, **spec** (A | Definition[A] | Iterable[A] | Iterable[Definition[A]]), **cast_to** (type[int]) → int | None

- **self**, **spec** (A | Definition[A] | Iterable[A] | Iterable[Definition[A]]), **cast_to** (type[bytes]) → bytes | None

**add_register_use**(*reg_offset*, *size*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_register_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_stack_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_stack_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_heap_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)

- **expr** (*Any | None*)

**add_heap_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_memory_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_memory_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_tmp_use**(*atom*, *code_loc*)

**Return type**
None

**Parameters**

- **atom** (Tmp)
- **code_loc** (CodeLocation)

**add_tmp_use_by_def**(*def_*, *code_loc*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)

**deref**(*pointer*, *size*, *endness=Endness.BE*)

**Overloads**

- **self**, **pointer** (MultiValues[claripy.ast.BV] | Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]), **size** (int | DerefSize), **endness** (archinfo.Endness) → set[MemoryLocation]
- **self**, **pointer** (int | claripy.ast.BV | HeapAddress | SpOffset), **size** (int | DerefSize), **endness** (archinfo.Endness) → MemoryLocation | None

**static is_heap_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**static get_heap_offset**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**heap_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int | HeapAddress*)

**class** angr.knowledge_plugins.key_definitions.**ReachingDefinitionsModel**

Bases: object

Models the definitions, uses, and memory of a ReachingDefinitionState object

**__init__**(*func_addr=None*, *track_liveness=True*)

**Parameters**

- **func_addr** (*int | None*)
- **track_liveness** (*bool*)

**add_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

**kill_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

**at_new_stmt**(*codeloc*)

**Return type**
None

**Parameters**
**codeloc** (CodeLocation)

**at_new_block**(*code_loc*, *pred_codelocs*)

**Return type**
None

**Parameters**

- **code_loc** (CodeLocation)
- **pred_codelocs** (*list*[CodeLocation])

**make_liveness_snapshot**()

**Return type**
None

**find_defs_at**(*code_loc*, *op=ObservationPointType.OP_BEFORE*)

**Return type**
set[*Definition*]

**Parameters**

- **code_loc** (CodeLocation)
- **op** (*int*)

**get_defs**(*atom*, *code_loc*, *op*)

**Return type**
set[*Definition*]

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **op** (*int*)

**copy**()

**Return type**
*ReachingDefinitionsModel*

**merge**(*model*)

**Parameters**
**model** (ReachingDefinitionsModel)

**get_observation_by_insn**(*ins_addr*, *kind*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **ins_addr** (*int* | CodeLocation)
- **kind** (ObservationPointType)

**get_observation_by_node**(*node_addr*, *kind*, *node_idx=None*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **node_addr** (*int* | CodeLocation)
- **kind** (ObservationPointType)
- **node_idx** (*int* | *None*)

**get_observation_by_stmt**(*arg1*, *arg2*, *arg3=None*, ***, *block_idx=None*)

**Overloads**

- **self**, **codeloc** (CodeLocation), **kind** (ObservationPointType) → LiveDefinitions | None
- **self**, **node_addr** (int), **stmt_idx** (int), **kind** (ObservationPointType), **block_idx** (int | None)

**get_observation_by_exit**(*node_addr*, *stmt_idx*, *src_node_idx=None*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **node_addr** (*int*)
- **stmt_idx** (*int*)
- **src_node_idx** (*int* | *None*)

**class** angr.knowledge_plugins.key_definitions.**Uses**

Bases: object

Describes uses (including the use location and the use expression) for definitions.

**__init__**(*uses_by_definition=None*, *uses_by_location=None*)

**Parameters**

- **uses_by_definition** (DefaultChainMapCOW | *None*)
- **uses_by_location** (DefaultChainMapCOW | *None*)

**add_use**(*definition*, *codeloc*, *expr=None*)

Add a use for a given definition.

**Parameters**

- **definition** (*Definition*) – The definition that is used.
- **codeloc** (*CodeLocation*) – The code location where the use occurs.
- **expr** (Any | None) – The expression that uses the specified definition at this location.

**get_uses**(*definition*)

Retrieve the uses of a given definition.

**Parameters**
**definition** (*Definition*) – The definition for which we get the uses.

**Return type**
set[*CodeLocation*]

**get_uses_with_expr**(*definition*)

Retrieve the uses and the corresponding expressions of a given definition.

**Parameters**
**definition** (*Definition*) – The definition for which we get the uses and the corresponding expressions.

**Return type**
set[tuple[*CodeLocation*, Any | None]]

**remove_use**(*definition*, *codeloc*, *expr=None*)

Remove one use of a given definition.

**Parameters**

- **definition** (*Definition*) – The definition of which to remove the uses.
- **codeloc** (*CodeLocation*) – The code location where the use is.
- **expr** (Any | None) – The expression that uses the definition at the given location.

**Return type**
None

**Returns**
None

**remove_uses**(*definition*)

Remove all uses of a given definition.

**Parameters**
**definition** (*Definition*) – The definition of which to remove the uses.

**Returns**
None

**get_uses_by_location**(*codeloc*, *exprs=False*)

Retrieve all definitions that are used at a given location.

**Parameters**

- **codeloc** (*CodeLocation*) – The code location.
- **exprs** (*bool*)

**Return type**
set[*Definition*] | set[tuple[*Definition*, Any | None]]

**Returns**
A set of definitions that are used at the given location.

**get_uses_by_insaddr**(*ins_addr*, *exprs=False*)

Retrieve all definitions that are used at a given location specified by the instruction address.

**Parameters**

- **ins_addr** (int) – The instruction address.
- **exprs** (*bool*)

**Return type**
set[*Definition*] | set[tuple[*Definition*, Any | None]]

**Returns**
A set of definitions that are used at the given location.

**copy**()

Copy the instance.

**Return type**
*Uses*

**Returns**
Return a new <Uses> instance containing the same data.

**merge**(*other*)

Merge an instance of <Uses> into the current instance.

**Parameters**
**other** (*Uses*) – The other <Uses> from which the data will be added to the current instance.

**Return type**
bool

**Returns**
True if any merge occurred, False otherwise

### Submodules

| | |
|---|---|
| *atoms* | |
| *constants* | |
| *definition* | |
| *environment* | |
| *heap_address* | |
| *key_definition_manager* | |
| *live_definitions* | |
| *liveness* | |
| *rd_model* | |
| *tag* | Classes to structure the different types of <Tag>s that can be attached to <Definition>s. |
| *undefined* | |
| *unknown_size* | |
| *uses* | |

### angr.knowledge_plugins.key_definitions.atoms

**class** angr.knowledge_plugins.key_definitions.atoms.**AtomKind**

Bases: Enum

An enum indicating the class of an atom

**REGISTER = 1**

**MEMORY = 2**

**TMP = 3**

**GUARD = 4**

**CONSTANT = 5**

class angr.knowledge_plugins.key_definitions.atoms.**Atom**

Bases: object

This class represents a data storage location manipulated by IR instructions.

It could either be a Tmp (temporary variable), a Register, a MemoryLocation.

**__init__**(*size*)

**Parameters**
**size** – The size of the atom in bytes

**size**

**property bits: int**

**static from_ail_expr**(*expr*, *arch*, *full_reg=False*)

**Return type**
*Register*

**Parameters**

- **expr** (Expression)
- **arch** (*Arch*)
- **full_reg** (*bool*)

**static from_argument**(*argument*, *arch*, *full_reg=False*, *sp=None*)

Instantiate an *Atom* from a given argument.

**Parameters**

- **argument** (*SimFunctionArgument*) – The argument to create a new atom from.
- **arch** (Arch) – The argument representing archinfo architecture for argument.
- **full_reg** – Whether to return an atom indicating the entire register if the argument only specifies a slice of the register.
- **sp** (int | None) – The current stack offset. Optional. Only used when argument is a SimStackArg.

**Return type**
*Register* | *MemoryLocation*

**static reg**(*thing*, *size=None*, *arch=None*)

Create a Register atom.

**Parameters**

- **thing** (str | RegisterOffset) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").
- **size** (int | None) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Arch | None) – The architecture. Must be provided when creating the atom using a register name.

**Return type**
*Register*

**Returns**
The Register Atom object.

**static register**(*thing*, *size=None*, *arch=None*)

Create a Register atom.

**Parameters**

- **thing** (str | RegisterOffset) – The register offset (e.g., project.arch.registers["rax"][0]) or the register name (e.g., "rax").
- **size** (int | None) – Size of the register atom. Must be provided when creating the atom using a register offset.
- **arch** (Arch | None) – The architecture. Must be provided when creating the atom using a register name.

**Return type**
*Register*

**Returns**
The Register Atom object.

**static mem**(*addr*, *size*, *endness=None*)

Create a MemoryLocation atom,

**Parameters**

- **addr** (*SpOffset* | *HeapAddress* | int) – The memory location. Can be an SpOffset for stack variables, an int for global memory variables, or a HeapAddress for items on the heap.
- **size** (int) – Size of the atom.
- **endness** (Endness | None) – Optional, either "Iend_LE" or "Iend_BE".

**Return type**
*MemoryLocation*

**Returns**
The MemoryLocation Atom object.

**static memory**(*addr*, *size*, *endness=None*)

Create a MemoryLocation atom,

**Parameters**

- **addr** (*SpOffset* | *HeapAddress* | int) – The memory location. Can be an SpOffset for stack variables, an int for global memory variables, or a HeapAddress for items on the heap.
- **size** (int) – Size of the atom.
- **endness** (Endness | None) – Optional, either "Iend_LE" or "Iend_BE".

**Return type**
*MemoryLocation*

**Returns**
The MemoryLocation Atom object.

**class** angr.knowledge_plugins.key_definitions.atoms.**GuardUse**

Bases: *Atom*

Implements a guard use.

**target**

**class** angr.knowledge_plugins.key_definitions.atoms.**ConstantSrc**

Bases: *Atom*

Represents a constant.

**value:** int

**class** angr.knowledge_plugins.key_definitions.atoms.**Tmp**

Bases: *Atom*

Represents a variable used by the IR to store intermediate values.

**tmp_idx**

**class** angr.knowledge_plugins.key_definitions.atoms.**Register**

Bases: *Atom*

Represents a given CPU register.

As an IR abstracts the CPU design to target different architectures, registers are represented as a separated memory space. Thus a register is defined by its offset from the base of this memory and its size.

**Variables**

- **reg_offset** (*int*) – The offset from the base to define its place in the memory bloc.
- **size** (*int*) – The size, in number of bytes.

**reg_offset**

**arch**

**property name:** str

**class** angr.knowledge_plugins.key_definitions.atoms.**VirtualVariable**

Bases: *Atom*

Represents a virtual variable.

**varid**

**category**

**oident**

**property was_reg:** bool

**property was_stack:** bool

**property was_parameter:** bool

**property was_tmp:** bool

**property reg_offset:** int | None

**property stack_offset:** int | None

**property tmp_idx:** int | None

class angr.knowledge_plugins.key_definitions.atoms.**MemoryLocation**

Bases: *Atom*

Represents a memory slice.

It is characterized by its address and its size.

**__init__**(*addr*, *size*, *endness=None*)

**Parameters**

- **addr** (*SpOffset* | *HeapAddress* | int) – The address of the beginning memory location slice.
- **size** (int) – The size of the represented memory location, in bytes.
- **endness** (*Endness* | *None*)

**addr:** *SpOffset* | int | BV

**endness**

**property is_on_stack:** bool

True if this memory location is located on the stack.

**property symbolic:** bool

### angr.knowledge_plugins.key_definitions.constants

class angr.knowledge_plugins.key_definitions.constants.**ObservationPointType**

Bases: IntEnum

Enum to replace the previously generic constants This makes it possible to annotate where they are expected by typing something as ObservationPointType instead of Literal[0,1]

**OP_BEFORE = 0**

**OP_AFTER = 1**

**__new__**(*value*)

### angr.knowledge_plugins.key_definitions.definition

class angr.knowledge_plugins.key_definitions.definition.**DefinitionMatchPredicate**

Bases: object

A dataclass indicating several facts which much all must match in order for a definition to match. Largely an internal class; don't worry about this.

**kind:** *AtomKind* | *type*[*Atom*] | None = None

**bbl_addr:** int | None = None

**ins_addr:** int | None = None

**variable:** *SimVariable* | None = None

**variable_manager:** *VariableManagerInternal* | Literal[False] | None = None

**stack_offset:** int | None = None

**reg_name:** str | int | None = None

**heap_offset:** int | None = None

**global_addr:** int | None = None

**tmp_idx:** int | None = None

**const_val:** int | None = None

**extern:** bool | None = None

**static construct**(*predicate=None*, ***kwargs*)

**Return type**
DefinitionMatchPredicate

**Parameters**
**predicate** (DefinitionMatchPredicate | None)

**normalize**()

**matches**(*defn*)

**Return type**
bool

**Parameters**
**defn** (Definition)

**__init__**(*kind=None*, *bbl_addr=None*, *ins_addr=None*, *variable=None*, *variable_manager=None*, *stack_offset=None*, *reg_name=None*, *heap_offset=None*, *global_addr=None*, *tmp_idx=None*, *const_val=None*, *extern=None*)

**Parameters**

- **kind** (AtomKind | type[Atom] | None)
- **bbl_addr** (int | None)
- **ins_addr** (int | None)
- **variable** (SimVariable | None)
- **variable_manager** (VariableManagerInternal | Literal[False] | None)
- **stack_offset** (int | None)
- **reg_name** (str | int | None)
- **heap_offset** (int | None)
- **global_addr** (int | None)
- **tmp_idx** (int | None)
- **const_val** (int | None)
- **extern** (bool | None)

**Return type**
None

class angr.knowledge_plugins.key_definitions.definition.**Definition**

Bases: Generic

An atom definition.

**Variables**

- **atom** – The atom being defined.
- **codeloc** – Where this definition is created in the original binary code.
- **dummy** – Tell whether the definition should be considered dummy or not. During simplification by AILment, definitions marked as dummy will not be removed.
- **tags** – A set of tags containing information about the definition gathered during analyses.

**__init__**(*atom*, *codeloc*, *dummy=False*, *tags=None*)

**Parameters**

- **atom** (*A*)
- **codeloc** (*CodeLoc*)
- **dummy** (*bool*)
- **tags** (*set[Tag] | None*)

**atom**

**codeloc**

**dummy**

**tags**

**property offset: int**

**property size: int**

**matches**(***kwargs*)

Return whether this definition has certain characteristics.

**Return type**
bool

### angr.knowledge_plugins.key_definitions.environment

class angr.knowledge_plugins.key_definitions.environment.**Environment**

Bases: object

Represent the environment in which a program runs. It's a mapping of variable names, to *claripy.ast.Base* that should contain possible addresses, or <UNDEFINED>, at which their respective values are stored.

**Note**: The <Environment> object does not store the values associated with variables themselves.

**__init__**(*environment=None*)

**Parameters**
**environment** (*dict[str | Undefined, set[Base]] | None*)

get(*names*)

**Parameters**
**names** (*set*[str]) – Potential values for the name of the environment variable to get the pointers of.

**Return type**
tuple[*set*[Base], bool]

**Returns**
The potential addresses of the values the environment variable can take; And a boolean value telling whether all the names were known of the internal representation (i.e. will be False if one of the queried variable was not found).

set(*name*, *pointers*)

**Parameters**

- **name** (str | *Undefined*) – Name of the environment variable to which we will associate the pointers.
- **pointers** (*set*[Base]) – New addresses where the new values of the environment variable are located.

merge(**others*)

**Return type**
tuple[*Environment*, bool]

**Parameters**
**others** (Environment)

compare(*other*)

**Return type**
bool

**Parameters**
**other** (Environment)

### angr.knowledge_plugins.key_definitions.heap_address

class angr.knowledge_plugins.key_definitions.heap_address.**HeapAddress**

Bases: object

The representation of an address on the heap.

__init__(*value*)

**Parameters**
**value** (*int* | Undefined)

property value

### angr.knowledge_plugins.key_definitions.key_definition_manager

class angr.knowledge_plugins.key_definitions.key_definition_manager.**RDAObserverControl**

Bases: object

**__init__**(*func_addr*, *call_site_block_addrs*, *call_site_ins_addrs*)

**Parameters**

- **func_addr** (*int*)
- **call_site_block_addrs** (*Iterable[int]*)
- **call_site_ins_addrs** (*Iterable[int]*)

**rda_observe_callback**(*ob_type*, ***kwargs*)

**class** angr.knowledge_plugins.key_definitions.key_definition_manager.**KeyDefinitionManager**

Bases: *KnowledgeBasePlugin*

KeyDefinitionManager manages and caches reaching definition models for each function.

For each function, by default we cache the entire reaching definitions model with observed results at the following locations: - Before each call instruction: ('insn', address of the call instruction, OP_BEFORE) - After returning from each call: ('node', address of the block that ends with a call, OP_AFTER)

**__init__**(*kb*)

**Parameters**
**kb** (KnowledgeBase)

**has_model**(*func_addr*)

**Parameters**
**func_addr** (*int*)

**get_model**(*func_addr*)

**Parameters**
**func_addr** (*int*)

**copy**()

**Return type**
*KeyDefinitionManager*

### angr.knowledge_plugins.key_definitions.live_definitions

**class** angr.knowledge_plugins.key_definitions.live_definitions.**DerefSize**

Bases: Enum

An enum for specialized kinds of dereferences

**NULL_TERMINATE - Dereference until the first byte which could be a literal null. Return a value including the**
terminator.

**NULL_TERMINATE = 1**

**class** angr.knowledge_plugins.key_definitions.live_definitions.**DefinitionAnnotation**

Bases: Annotation

An annotation that attaches a *Definition* to an AST.

**relocatable = True**

**eliminatable = False**

__init__(*definition*)

definition

class angr.knowledge_plugins.key_definitions.live_definitions.**LiveDefinitions**

Bases: object

A LiveDefinitions instance contains definitions and uses for register, stack, memory, and temporary variables, uncovered during the analysis.

INITIAL_SP_16BIT = 32512

INITIAL_SP_32BIT = 2147418112

INITIAL_SP_64BIT = 140737488289792

__init__(*arch*, *track_tmps=False*, *canonical_size=8*, *registers=None*, *stack=None*, *memory=None*, *heap=None*, *tmps=None*, *others=None*, *register_uses=None*, *stack_uses=None*, *heap_uses=None*, *memory_uses=None*, *tmp_uses=None*, *other_uses=None*, *element_limit=5*, *merge_into_tops=True*)

**Parameters**

- **arch** (*Arch*)
- **track_tmps** (*bool*)
- **merge_into_tops** (*bool*)

project: *Project* | None

arch

track_tmps

registers: *MultiValuedMemory*

stack: *MultiValuedMemory*

memory: *MultiValuedMemory*

heap: *MultiValuedMemory*

tmps: dict[int, set[*Definition*]]

others: dict[*Atom*, *MultiValues*]

register_uses

stack_uses

heap_uses

memory_uses

tmp_uses: dict[int, set[*CodeLocation*]]

other_uses

uses_by_codeloc: dict[*CodeLocation*, set[*Definition*]]

**copy**(*discard_tmpdefs=False*)

**Return type**
*LiveDefinitions*

**reset_uses**()

**static top**(*bits*)

Get a TOP value.

**Parameters**
**bits** (int) – Width of the TOP value (in bits).

**Returns**
The TOP value.

**static is_top**(*expr*)

Check if the given expression is a TOP value.

**Parameters**
**expr** – The given expression.

**Return type**
bool

**Returns**
True if the expression is TOP, False otherwise.

**stack_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int*)

**static is_stack_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**static get_stack_offset**(*addr*, *had_stack_base=False*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**static annotate_with_def**(*symvar*, *definition*)

**Parameters**

- **symvar** (TypeVar(MVType, bound= BV | FP))
- **definition** (*Definition*)

**Return type**
TypeVar(MVType, bound= BV | FP)

**Returns**

static extract_defs(*symvar*)

**Return type**
Generator[*Definition*]

**Parameters**
**symvar** (*Base*)

static extract_defs_from_annotations(*annos*)

**Return type**
set[*Definition*]

**Parameters**
**annos** (*Iterable[Annotation]*)

static extract_defs_from_mv(*mv*)

**Return type**
Generator[*Definition*]

**Parameters**
**mv** (MultiValues)

get_sp()

Return the concrete value contained by the stack pointer.

**Return type**
int

get_sp_offset()

Return the offset of the stack pointer.

**Return type**
int | None

get_stack_address(*offset*)

**Return type**
int | None

**Parameters**
**offset** (*Base*)

stack_offset_to_stack_addr(*offset*)

**Return type**
int

merge(**others*)

**Return type**
tuple[*LiveDefinitions*, bool]

**Parameters**
**others** (LiveDefinitions)

compare(*other*)

**Return type**
bool

**Parameters**
**other** (LiveDefinitions)

**kill_definitions**(*atom*)

Overwrite existing definitions w.r.t 'atom' with a dummy definition instance. A dummy definition will not be removed during simplification.

**Parameters**
**atom** (*Atom*)

**Return type**
None

**Returns**
None

**kill_and_add_definition**(*atom*, *code_loc*, *data*, *dummy=False*, *tags=None*, *endness=None*, *annotated=False*)

**Return type**
*MultiValues* | None

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **data** (MultiValues)
- **tags** (*set[Tag] | None*)

**add_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_use_by_def**(*definition*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **definition** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any*)

**get_definitions**(*thing*)

**get_tmp_definitions**(*tmp_idx*)

**Return type**
set[*Definition*]

**Parameters**
**tmp_idx** (*int*)

**get_register_definitions**(*reg_offset*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)

**get_stack_values**(*stack_offset*, *size*, *endness*)

**Return type**
*MultiValues* | None

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)
- **endness** (*str*)

**get_stack_definitions**(*stack_offset*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **stack_offset** (*int*)
- **size** (*int*)

**get_heap_definitions**(*heap_addr*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **heap_addr** (*int*)
- **size** (*int*)

**get_memory_definitions**(*addr*, *size*)

**Return type**
set[*Definition*]

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**get_values**(*spec*, *endness=None*)

**get_one_value**(*spec*, *strip_annotations=False*)

get_concrete_value(*spec*, *cast_to=<class 'int'>*)

**Overloads**

- **self**, **spec** (A | Definition[A] | Iterable[A] | Iterable[Definition[A]]), **cast_to** (type[int]) → int | None
- **self**, **spec** (A | Definition[A] | Iterable[A] | Iterable[Definition[A]]), **cast_to** (type[bytes]) → bytes | None

add_register_use(*reg_offset*, *size*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **reg_offset** (*int*)
- **size** (*int*)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

add_register_use_by_def(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

add_stack_use(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

add_stack_use_by_def(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_heap_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_heap_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_memory_use**(*atom*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **atom** (MemoryLocation)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_memory_use_by_def**(*def_*, *code_loc*, *expr=None*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)
- **expr** (*Any | None*)

**add_tmp_use**(*atom*, *code_loc*)

**Return type**
None

**Parameters**

- **atom** (Tmp)
- **code_loc** (CodeLocation)

**add_tmp_use_by_def**(*def_*, *code_loc*)

**Return type**
None

**Parameters**

- **def_** (Definition)
- **code_loc** (CodeLocation)

**deref**(*pointer*, *size*, *endness=Endness.BE*)

**Overloads**

- **self**, **pointer** (MultiValues[claripy.ast.BV] | Atom | Definition[Atom] | Iterable[Atom] | Iterable[Definition[Atom]]), **size** (int | DerefSize), **endness** (archinfo.Endness) → set[MemoryLocation]
- **self**, **pointer** (int | claripy.ast.BV | HeapAddress | SpOffset), **size** (int | DerefSize), **endness** (archinfo.Endness) → MemoryLocation | None

**static is_heap_address**(*addr*)

**Return type**
bool

**Parameters**
**addr** (*Base*)

**static get_heap_offset**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*Base*)

**heap_address**(*offset*)

**Return type**
BV

**Parameters**
**offset** (*int | HeapAddress*)

### angr.knowledge_plugins.key_definitions.liveness

**class** angr.knowledge_plugins.key_definitions.liveness.**Liveness**

Bases: object

This class stores liveness information for each definition.

**__init__**()

**add_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

**kill_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

make_liveness_snapshot()

**Return type**
None

at_new_stmt(*code_loc*)

Only support moving from a statement to the next statement within one basic block.

**Return type**
None

**Parameters**
**code_loc** (CodeLocation)

at_new_block(*code_loc*, *pred_codelocs*)

Only support moving to a new block from one or more blocks.

**Return type**
None

**Parameters**

- **code_loc** (CodeLocation)
- **pred_codelocs** (*list*[CodeLocation])

find_defs_at(*code_loc*, *op=ObservationPointType.OP_BEFORE*)

**Return type**
set[*Definition*]

**Parameters**

- **code_loc** (CodeLocation)
- **op** (*int*)

find_defs_at_raw(*block_addr*, *block_idx*, *stmt_idx*, *op=ObservationPointType.OP_BEFORE*)

**Return type**
set[*Definition*]

**Parameters**

- **block_addr** (*int*)
- **block_idx** (*int | None*)
- **stmt_idx** (*int | None*)
- **op** (*int*)

copy()

**Return type**
*Liveness*

### angr.knowledge_plugins.key_definitions.rd_model

**class** angr.knowledge_plugins.key_definitions.rd_model.**ReachingDefinitionsModel**

Bases: object

Models the definitions, uses, and memory of a ReachingDefinitionState object

**__init__**(*func_addr=None*, *track_liveness=True*)

**Parameters**

- **func_addr** (*int | None*)
- **track_liveness** (*bool*)

**add_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

**kill_def**(*d*)

**Return type**
None

**Parameters**
**d** (Definition)

**at_new_stmt**(*codeloc*)

**Return type**
None

**Parameters**
**codeloc** (CodeLocation)

**at_new_block**(*code_loc*, *pred_codelocs*)

**Return type**
None

**Parameters**

- **code_loc** (CodeLocation)
- **pred_codelocs** (*list[CodeLocation]*)

**make_liveness_snapshot**()

**Return type**
None

**find_defs_at**(*code_loc*, *op=ObservationPointType.OP_BEFORE*)

**Return type**
set[*Definition*]

**Parameters**

- **code_loc** (CodeLocation)
- **op** (*int*)

**get_defs**(*atom*, *code_loc*, *op*)

**Return type**
set[*Definition*]

**Parameters**

- **atom** (Atom)
- **code_loc** (CodeLocation)
- **op** (*int*)

**copy**()

**Return type**
*ReachingDefinitionsModel*

**merge**(*model*)

**Parameters**
**model** (ReachingDefinitionsModel)

**get_observation_by_insn**(*ins_addr*, *kind*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **ins_addr** (*int* | CodeLocation)
- **kind** (ObservationPointType)

**get_observation_by_node**(*node_addr*, *kind*, *node_idx=None*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **node_addr** (*int* | CodeLocation)
- **kind** (ObservationPointType)
- **node_idx** (*int* | *None*)

**get_observation_by_stmt**(*arg1*, *arg2*, *arg3=None*, ***, *block_idx=None*)

**Overloads**

- **self**, **codeloc** (CodeLocation), **kind** (ObservationPointType) → LiveDefinitions | None
- **self**, **node_addr** (int), **stmt_idx** (int), **kind** (ObservationPointType), **block_idx** (int | None)

**get_observation_by_exit**(*node_addr*, *stmt_idx*, *src_node_idx=None*)

**Return type**
*LiveDefinitions* | None

**Parameters**

- **node_addr** (*int*)
- **stmt_idx** (*int*)
- **src_node_idx** (*int* | *None*)

### angr.knowledge_plugins.key_definitions.tag

Classes to structure the different types of <Tag>s that can be attached to <Definition>s.

- **Tag**
  - **FunctionTag**
    - ParameterTag
    - LocalVariableTag
    - ReturnValueTag
  - InitialValueTag

**class** angr.knowledge_plugins.key_definitions.tag.**Tag**

Bases: object

A tag for a Definition that can carry different kinds of metadata.

**__init__**(*metadata=None*)

**Parameters**
**metadata** (*object*)

**class** angr.knowledge_plugins.key_definitions.tag.**FunctionTag**

Bases: *Tag*

A tag for a definition created (or used) in the context of a function.

**__init__**(*function=None*, *metadata=None*)

**Parameters**

- **function** (*int | None*)
- **metadata** (*object*)

**class** angr.knowledge_plugins.key_definitions.tag.**SideEffectTag**

Bases: *FunctionTag*

A tag for a definition created or used as a side-effect of a function.

Example: The <MemoryLocation> pointed by *rdi* during a *sprintf*.

**class** angr.knowledge_plugins.key_definitions.tag.**ParameterTag**

Bases: *FunctionTag*

A tag for a definition of a parameter.

**class** angr.knowledge_plugins.key_definitions.tag.**LocalVariableTag**

Bases: *FunctionTag*

A tag for a definition of a local variable of a function.

**class** angr.knowledge_plugins.key_definitions.tag.**ReturnValueTag**

Bases: *FunctionTag*

A tag for a definition of a return value of a function.

**class** angr.knowledge_plugins.key_definitions.tag.**InitialValueTag**

Bases: *Tag*

A tag for a definition of an initial value

class angr.knowledge_plugins.key_definitions.tag.**UnknownSizeTag**

Bases: *Tag*

A tag for a definition of an initial value

### angr.knowledge_plugins.key_definitions.undefined

class angr.knowledge_plugins.key_definitions.undefined.**Undefined**

Bases: object

A TOP-like value indicating an unknown data source. Should live next to raw integers in DataSets.

### angr.knowledge_plugins.key_definitions.unknown_size

class angr.knowledge_plugins.key_definitions.unknown_size.**UnknownSize**

Bases: object

A value indicating an unknown size for elements of DataSets. Should "behave" like an integer.

### angr.knowledge_plugins.key_definitions.uses

class angr.knowledge_plugins.key_definitions.uses.**Uses**

Bases: object

Describes uses (including the use location and the use expression) for definitions.

**__init__**(*uses_by_definition=None*, *uses_by_location=None*)

**Parameters**

- **uses_by_definition** (DefaultChainMapCOW | *None*)
- **uses_by_location** (DefaultChainMapCOW | *None*)

**add_use**(*definition*, *codeloc*, *expr=None*)

Add a use for a given definition.

**Parameters**

- **definition** (*Definition*) – The definition that is used.
- **codeloc** (*CodeLocation*) – The code location where the use occurs.
- **expr** (Any | None) – The expression that uses the specified definition at this location.

**get_uses**(*definition*)

Retrieve the uses of a given definition.

**Parameters**
**definition** (*Definition*) – The definition for which we get the uses.

**Return type**
set[*CodeLocation*]

**get_uses_with_expr**(*definition*)

Retrieve the uses and the corresponding expressions of a given definition.

**Parameters**
**definition** (*Definition*) – The definition for which we get the uses and the corresponding expressions.

**Return type**
set[tuple[*CodeLocation*, Any | None]]

**remove_use**(*definition*, *codeloc*, *expr=None*)

Remove one use of a given definition.

**Parameters**

- **definition** (*Definition*) – The definition of which to remove the uses.
- **codeloc** (*CodeLocation*) – The code location where the use is.
- **expr** (Any | None) – The expression that uses the definition at the given location.

**Return type**
None

**Returns**
None

**remove_uses**(*definition*)

Remove all uses of a given definition.

**Parameters**
**definition** (*Definition*) – The definition of which to remove the uses.

**Returns**
None

**get_uses_by_location**(*codeloc*, *exprs=False*)

Retrieve all definitions that are used at a given location.

**Parameters**

- **codeloc** (*CodeLocation*) – The code location.
- **exprs** (*bool*)

**Return type**
set[*Definition*] | set[tuple[*Definition*, Any | None]]

**Returns**
A set of definitions that are used at the given location.

**get_uses_by_insaddr**(*ins_addr*, *exprs=False*)

Retrieve all definitions that are used at a given location specified by the instruction address.

**Parameters**

- **ins_addr** (int) – The instruction address.
- **exprs** (*bool*)

**Return type**
set[*Definition*] | set[tuple[*Definition*, Any | None]]

**Returns**
A set of definitions that are used at the given location.

**copy**()

Copy the instance.

**Return type**
*Uses*

**Returns**
Return a new <Uses> instance containing the same data.

**merge**(*other*)

Merge an instance of <Uses> into the current instance.

**Parameters**
**other** (*Uses*) – The other <Uses> from which the data will be added to the current instance.

**Return type**
bool

**Returns**
True if any merge occurred, False otherwise

### 10.22.11 angr.knowledge_plugins.labels

**class** angr.knowledge_plugins.labels.**Labels**

Bases: *KnowledgeBasePlugin*

**__init__**(*kb*)

**items**()

**get**(*addr*, *default=None*)

Get a label as string for a given address Same as .labels[x]

**lookup**(*name*, *default=None*)

Returns an address to a given label To show all available labels, iterate over .labels or list(b.kb.labels)

**copy**()

**get_unique_label**(*label*)

Get a unique label name from the given label name.

**Parameters**
**label** (*str*) – The desired label name.

**Returns**
A unique label name.

### 10.22.12 angr.knowledge_plugins.obfuscations

**class** angr.knowledge_plugins.obfuscations.**Obfuscations**

Bases: *KnowledgeBasePlugin*

Store discovered information and artifacts about (string) obfuscation techniques in the project.

**__init__**(*kb*)

**copy**()

### 10.22.13 angr.knowledge_plugins.patches

**class** angr.knowledge_plugins.patches.**Patch**

Bases: object

**__init__**(*addr*, *new_bytes*, *comment=None*)

**Parameters**
**comment** (*str | None*)

**class** angr.knowledge_plugins.patches.**PatchManager**

Bases: *KnowledgeBasePlugin*

A placeholder-style implementation for a binary patch manager. This class should be significantly changed in the future when all data about loaded binary objects are loaded into angr knowledge base from CLE. As of now, it only stores byte-level replacements.

Patches should not overlap, but it's user's responsibility to check for and avoid overlapping patches.

**__init__**(*kb*)

**add_patch**(*addr*, *new_bytes*, *comment=None*)

**Parameters**
**comment** (*str | None*)

**add_patch_obj**(*patch*)

**Parameters**
**patch** (Patch)

**remove_patch**(*addr*)

**patch_addrs**()

**get_patch**(*addr*)

Get patch at the given address.

**Parameters**
**addr** (*int*) – The address of the patch.

**Returns**
The patch if there is one starting at the address, or None if there isn't any.

**Return type**
*Patch* or None

**get_all_patches**(*addr*, *size*)

Retrieve all patches that cover a region specified by [addr, addr+size).

**Parameters**

- **addr** (*int*) – The address of the beginning of the region.
- **size** (*int*) – Size of the region.

**Returns**
A list of patches.

**Return type**
list

**keys**()

**items**()

**values**()

**copy**()

**static overlap**(*a0*, *a1*, *b0*, *b1*)

**apply_patches_to_binary**(*binary_bytes=None*, *patches=None*)

**Return type**
bytes

**Parameters**

- **binary_bytes** (bytes | *None*)
- **patches** (*list[Patch]* | *None*)

**apply_patches_to_state**(*state*)

**property patched_entry_state**

### 10.22.14 angr.knowledge_plugins.plugin

**class** angr.knowledge_plugins.plugin.**KnowledgeBasePlugin**

Bases: object

**__init__**(*kb*)

**Parameters**
**kb** (KnowledgeBase)

**copy**()

**set_kb**(*kb*)

Invoked after __setstate__() is called.

**Parameters**
**kb** (KnowledgeBase)

**static register_default**(*name*, *cls*)

### 10.22.15 angr.knowledge_plugins.propagations

**class** angr.knowledge_plugins.propagations.**PropagationManager**

Bases: *KnowledgeBasePlugin*

Manages the results of Propagator, including intermediate results for unfinished Propagation runs.

**__init__**(*kb*)

**exists**(*prop_key*)

Internal function to check if a func, specified as a CodeLocation exists in our known propagations

**Parameters**
**prop_key** (tuple) – A key of the propagation result.

**Return type**
bool

**Returns**
Whether such a key exists or not.

**update**(*prop_key*, *model*)

Add the replacements to known propagations

**Parameters**

- **prop_key** (tuple) – A key of the propagation result.

- **model** (*PropagationModel*) – The propagation result to store

**Return type**
None

**get**(*prop_key*, *default=None*)

Gets the replacements for a specified function location. If the replacement does not exist in the known propagations, it returns None.

**Parameters**

- **prop_key** – A key of the propagation result.
- **default** – The default value to return if the prop_key does not exist in the cache.

**Return type**
*PropagationModel*

**Returns**
Dict or None

**copy**()

**discard_by_prefix**(*prefix*)

**Parameters**
**prefix** (*str*)

**class** angr.knowledge_plugins.propagations.**PropagationModel**

Bases: *Serializable*

This class stores the propagation result that comes out of Propagator.

**__init__**(*prop_key*, *node_iterations=None*, *states=None*, *block_initial_reg_values=None*, *replacements=None*, *equivalence=None*, *function=None*, *input_states=None*, *function_block_count=None*)

**Parameters**

- **prop_key** (*tuple*)
- **node_iterations** (*defaultdict[Any, int] | None*)
- **states** (*dict | None*)
- **block_initial_reg_values** (*dict | None*)
- **replacements** (*defaultdict[Any, dict] | None*)
- **equivalence** (*set | None*)
- **function** (*Function | None*)
- **input_states** (*dict | None*)
- **function_block_count** (*int | None*)

**key**

**node_iterations**

**input_states**

**states**

**block_initial_reg_values**

**replacements**

**equivalence**

**graph_visitor**

**function_block_count**

**downsize**()

**block_beginning_state**(*block_addr*)

**Return type**
*PropagatorState*

## Submodules


*prop_value*
*propagation_manager*
*propagation_model*
*states*


### angr.knowledge_plugins.propagations.prop_value

**class** angr.knowledge_plugins.propagations.prop_value.**Detail**

Bases: object

A companion class used together with PropValue. It describes stored information at each offset (in bytes).

**Variables**
**def_at** – Where this expression is defined, or None if it was never explicitly defined in the current block or the current function.

**__init__**(*size*, *expr*, *def_at*)

**Parameters**

- **size** (*int*)
- **expr** (Expression | *None*)
- **def_at** (CodeLocation | *None*)

**size**

**expr**

**def_at**

**class** angr.knowledge_plugins.propagations.prop_value.**PropValue**

Bases: object

Describes immutable basic value type that is used in Propagator.

**__init__**(*value*, *offset_and_details=None*)

**Parameters**

- **value** (*Bits*)
- **offset_and_details** (*dict[int, Detail] | None*)

**value**

**offset_and_details**

**property needs_details**

**property one_expr: *Expression* | None**

Get the expression that starts at offset 0 and covers the entire PropValue. Returns None if there are no expressions or multiple expressions.

**property one_defat: *CodeLocation* | None**

Get the definition location of the expression that starts at offset 0 and covers the entire PropValue. Returns None if there are no expressions or multiple expressions.

**to_label**()

**with_details**(*size*, *expr*, *def_at*)

**Return type**
*PropValue*

**Parameters**

- **size** (*int*)
- **expr** (Expression)
- **def_at** (CodeLocation)

**all_exprs**()

**Return type**
Generator[*Expression*]

**non_zero_exprs**()

**Return type**
Generator[*Expression*]

**static chop_value**(*value*, *begin_offset*, *end_offset*)

**Return type**
Bits

**Parameters**
**value** (*Bits*)

**value_and_labels**()

**Return type**
Generator[tuple[int, Bits, int, dict | None]]

static **from_value_and_labels**(*value*, *labels*)

**Return type**
*PropValue*

**Parameters**

- **value** (*Bits*)
- **labels** (*Iterable[tuple[int, int, int, dict[str, Any]]]*)

static **from_value_and_details**(*value*, *size*, *expr*, *def_at*)

**Parameters**

- **value** (*Bits*)
- **size** (*int*)
- **expr** (Expression)
- **def_at** (CodeLocation)

static **extract_ail_expression**(*start*, *bits*, *expr*)

**Return type**
*Expression* | None

**Parameters**

- **start** (*int*)
- **bits** (*int*)
- **expr** (*Expression | None*)

static **extend_ail_expression**(*bits*, *expr*)

**Return type**
*Expression* | None

**Parameters**

- **bits** (*int*)
- **expr** (*Expression | None*)

### angr.knowledge_plugins.propagations.propagation_manager

class angr.knowledge_plugins.propagations.propagation_manager.**PropagationManager**

Bases: *KnowledgeBasePlugin*

Manages the results of Propagator, including intermediate results for unfinished Propagation runs.

**__init__**(*kb*)

**exists**(*prop_key*)

Internal function to check if a func, specified as a CodeLocation exists in our known propagations

**Parameters**
**prop_key** (tuple) – A key of the propagation result.

**Return type**
bool

**Returns**
Whether such a key exists or not.

**update**(*prop_key*, *model*)

Add the replacements to known propagations

**Parameters**

- **prop_key** (tuple) – A key of the propagation result.
- **model** (*PropagationModel*) – The propagation result to store

**Return type**
None

**get**(*prop_key*, *default=None*)

Gets the replacements for a specified function location. If the replacement does not exist in the known propagations, it returns None.

**Parameters**

- **prop_key** – A key of the propagation result.
- **default** – The default value to return if the prop_key does not exist in the cache.

**Return type**
*PropagationModel*

**Returns**
Dict or None

**copy**()

**discard_by_prefix**(*prefix*)

**Parameters**
**prefix** (*str*)

### angr.knowledge_plugins.propagations.propagation_model

**class** angr.knowledge_plugins.propagations.propagation_model.**PropagationModel**

Bases: *Serializable*

This class stores the propagation result that comes out of Propagator.

**__init__**(*prop_key*, *node_iterations=None*, *states=None*, *block_initial_reg_values=None*, *replacements=None*, *equivalence=None*, *function=None*, *input_states=None*, *function_block_count=None*)

**Parameters**

- **prop_key** (*tuple*)
- **node_iterations** (*defaultdict[Any, int] | None*)
- **states** (*dict | None*)
- **block_initial_reg_values** (*dict | None*)
- **replacements** (*defaultdict[Any, dict] | None*)
- **equivalence** (*set | None*)
- **function** (Function | *None*)

- **input_states** (*dict | None*)
- **function_block_count** (*int | None*)

**key**

**node_iterations**

**input_states**

**states**

**block_initial_reg_values**

**replacements**

**equivalence**

**graph_visitor**

**function_block_count**

**downsize**()

**block_beginning_state**(*block_addr*)

**Return type**
*PropagatorState*

### angr.knowledge_plugins.propagations.states

**class** angr.knowledge_plugins.propagations.states.**CallExprFinder**

Bases: *AILBlockRewriter*

Walks an AIL expression to find if it contains a call expression anywhere.

**__init__**()

**class** angr.knowledge_plugins.propagations.states.**PropagatorState**

Bases: object

Describes the base state used in Propagator.

**Variables**

- **arch** – Architecture of the binary.
- **gp** – value of the global pointer for MIPS binaries.
- **_replacements** – Stores expressions to replace, keyed by CodeLocation instances
- **_only_consts** – Only track constants.
- **_expr_used_locs** – A dict keyed by expressions and valued by CodeLocations where the expression is used.
- **_max_prop_expr_occurrence** – The upperbound for the number of occurrences of an expression for Propagator to propagate that expression to new locations (and replace the original expression). Setting it to 0 disables this limit, which means Propagator will always propagate expressions regardless of how many times it has been propagated.

__init__(*arch*, *project=None*, *replacements=None*, *only_consts=False*, *expr_used_locs=None*, *store_tops=True*, *gp=None*, *max_prop_expr_occurrence=1*, *model=None*, *artificial_reg_offsets=None*)

**Parameters**

- **arch** (*Arch*)
- **project** (*Project | None*)
- **replacements** (*defaultdict[CodeLocation, dict] | None*)
- **only_consts** (*bool*)
- **expr_used_locs** (*defaultdict[Any, set[CodeLocation]] | None*)
- **store_tops** (*bool*)
- **gp** (*int | None*)
- **max_prop_expr_occurrence** (*int*)

**arch**

**gpr_size**

**project**

**model**

**classmethod initial_state**(*project*, *only_consts=False*, *gp=None*, *do_binops=True*, *store_tops=False*, *func_addr=None*, *max_prop_expr_occurrence=None*, *initial_codeloc=None*)

**Parameters**
**project** (*Project*)

**static top**(*bits*)

Get a TOP value.

**Parameters**

- **size** – Width of the TOP value (in bits).
- **bits** (*int*)

**Return type**
BV

**Returns**
The TOP value.

**static is_top**(*expr*)

Check if the given expression is a TOP value.

**Parameters**
**expr** – The given expression.

**Return type**
bool

**Returns**
True if the expression is TOP, False otherwise.

**static merge_replacements**(*replacements_0*, *replacements_1*)

The replacement merging logic is special: replacements_1 is the newer replacement result and replacement_0 is the older result waiting to be updated. When both replacements_1 and replacement_0 have a non-top value for the same variable and code location, we will update the slot in replacement_0 with the value from replacement_1.

**Return type**
bool

**Returns**
Whether merging has happened or not.

**abstractmethod copy**()

**Return type**
Self

**merge**(**others*)

**Return type**
tuple[Self, bool]

**Parameters**
**others** (*Self*)

**init_replacements**()

**add_replacement**(*codeloc*, *old*, *new*, *force_replace=False*)

Add a replacement record: Replacing expression *old* with *new* at program location *codeloc*. If the self._only_consts flag is set to true, only constant values will be set.

**Parameters**

- **codeloc** (*CodeLocation*) – The code location.
- **old** – The expression to be replaced.
- **new** – The expression to replace with.
- **force_replace** (*bool*)

**Return type**
bool

**Returns**
True if the replacement will happen. False otherwise.

**filter_replacements**()

**has_replacements_at**(*codeloc*)

**Return type**
bool

**Parameters**
**codeloc** (CodeLocation)

**class** angr.knowledge_plugins.propagations.states.**RegisterAnnotation**

Bases: Annotation

Annotates TOP values that are coming from registers.

**eliminatable = True**

**relocatable = True**

**__init__**(*offset*, *size*)

**class** angr.knowledge_plugins.propagations.states.**RegisterComparisonAnnotation**

Bases: Annotation

Annotate TOP values that are the result of register values comparing against constant values.

**eliminatable = True**

**relocatable = True**

**__init__**(*offset*, *size*, *cmp_op*, *value*)

**Parameters**

- **offset** (*int*)
- **size** (*int*)
- **cmp_op** (*str*)
- **value** (*int*)

**class** angr.knowledge_plugins.propagations.states.**PropagatorVEXState**

Bases: *PropagatorState*

Describes the state used in the VEX engine of Propagator.

**__init__**(*arch*, *project=None*, *registers=None*, *local_variables=None*, *replacements=None*, *only_consts=False*, *expr_used_locs=None*, *do_binops=True*, *store_tops=True*, *block_initial_reg_values=None*, *gp=None*, *max_prop_expr_occurrence=1*, *model=None*, *artificial_reg_offsets=None*)

**Parameters**

**max_prop_expr_occurrence** (*int*)

**do_binops**

**block_initial_reg_values**

**classmethod initial_state**(*project*, *only_consts=False*, *gp=None*, *do_binops=True*, *store_tops=False*, *func_addr=None*, *max_prop_expr_occurrence=None*, *initial_codeloc=None*, *model=None*)

**copy**()

**Return type**

*PropagatorVEXState*

**merge**(**others*)

**Return type**

tuple[*PropagatorVEXState*, bool]

**Parameters**

**others** (PropagatorVEXState)

**store_local_variable**(*offset*, *size*, *value*, *endness*)

**load_local_variable**(*offset*, *size*, *endness*)

**Return type**
BV

**store_register**(*offset*, *size*, *value*)

**load_register**(*offset*, *size*)

**register_results**()

**Return type**
dict[str, BV]

class angr.knowledge_plugins.propagations.states.**Equivalence**

Bases: object

Describes an equivalence relationship between two atoms.

**__init__**(*codeloc*, *atom0*, *atom1*, *is_weakassignment=False*)

**Parameters**
**is_weakassignment** (*bool*)

**codeloc**

**atom0**

**atom1**

**is_weakassignment**

### 10.22.16 angr.knowledge_plugins.rtdb

class angr.knowledge_plugins.rtdb.**RuntimeDb**

Bases: *KnowledgeBasePlugin*

External storage-backed database for angr knowledge base plugins.

**__init__**(*kb*, *lmdb_path=None*)

**Parameters**
- **kb** (KnowledgeBase)
- **lmdb_path** (*str | None*)

**Return type**
None

**increase_lmdb_map_size**()

Increase the LMDB map size.

Note that the old database handle *may* no longer be valid after a map size increase. rhelmot could reproduce the error "Database handle belongs to another environment." in nix + CPython 3.13.13. Reopening all databases after increasing LMDB map size solves this issue.

**Return type**
None

**reopen_lmdb**()

Reopen the existing LMDB environment and all open databases in self._dbs.

**reopen_lmdb_databases**()

**open_db**(*db_name*, *unique=True*)

**Return type**
str

**Parameters**

- **db_name** (*str*)
- **unique** (*bool*)

**begin_txn**(*db_name*, *write=False*)

**Parameters**

- **db_name** (*str*)
- **write** (*bool*)

**drop_db**(*db_name*)

**Return type**
None

**Parameters**
**db_name** (*str*)

**cleanup**()

Close the LMDB environment and remove the rtdb directory if this is the last user of it. This method may be invoked multiple times (e.g., once by the atexit hook and once more by __del__) and should not fail.

## Submodules

*rtdb*

### angr.knowledge_plugins.rtdb.rtdb

**class** angr.knowledge_plugins.rtdb.rtdb.**RuntimeDbForkCondom**

Bases: object

A class that invokes RuntimeDb.reopen_lmdb() upon forking. This is necessary to ensure that lmdb does not raise ReaderFullError in forked child processes. The reopen_rtdb() method is not called on Windows because os.register_at_fork does not exist on Windows.

**__init__**(*rtdb*)

**Parameters**
**rtdb** (RuntimeDb)

**reopen_rtdb**()

**class** angr.knowledge_plugins.rtdb.rtdb.**RuntimeDb**

Bases: *KnowledgeBasePlugin*

External storage-backed database for angr knowledge base plugins.

__init__(*kb*, *lmdb_path=None*)

**Parameters**

- **kb** (KnowledgeBase)
- **lmdb_path** (*str | None*)

**Return type**
None

increase_lmdb_map_size()

Increase the LMDB map size.

Note that the old database handle *may* no longer be valid after a map size increase. rhelmot could reproduce the error "Database handle belongs to another environment." in nix + CPython 3.13.13. Reopening all databases after increasing LMDB map size solves this issue.

**Return type**
None

reopen_lmdb()

Reopen the existing LMDB environment and all open databases in self._dbs.

reopen_lmdb_databases()

open_db(*db_name*, *unique=True*)

**Return type**
str

**Parameters**

- **db_name** (*str*)
- **unique** (*bool*)

begin_txn(*db_name*, *write=False*)

**Parameters**

- **db_name** (*str*)
- **write** (*bool*)

drop_db(*db_name*)

**Return type**
None

**Parameters**
**db_name** (*str*)

cleanup()

Close the LMDB environment and remove the rtdb directory if this is the last user of it. This method may be invoked multiple times (e.g., once by the atexit hook and once more by __del__) and should not fail.

### 10.22.17 angr.knowledge_plugins.structured_code

**class** angr.knowledge_plugins.structured_code.**SpillingDecompilationDict**

Bases: MutableMapping

A dict of decompilation caches, keyed by (function address, flavor), that keeps only the most recently used cache_limit entries in memory and spills the rest to an LMDB database managed by the RuntimeDb knowledge base plugin.

Evicted entries are always serialized and written out (caches are mutated in place, so there is no clean/dirty distinction). Entries that cannot be serialized (e.g. DummyStructuredCodeGenerator or rust-flavor caches) are parked in an unbounded in-memory dict. Spilled entries are deserialized on access with the owning knowledge base's project/function attached.

**__init__**(*kb*, *cache_limit=128*)

**Parameters**

- **kb** (KnowledgeBase)
- **cache_limit** (*int*)

**property cache_limit: int**

**export_serialized**()

Export all serializable entries as (key, serialized bytes) pairs. Spilled entries are copied directly out of the LMDB backing store without being deserialized and re-serialized. Entries that cannot be serialized are returned separately in a dict.

**Return type**
tuple[list[tuple[tuple[int, str], bytes]], dict[tuple[int, str], *DecompilationCache*]]

**bulk_import_serialized**(*items*)

Move already-serialized decompilation caches directly into the LMDB backing store and register them as spilled, without deserializing them. The bytes must be serialized DecompilationCache messages, i.e., the exact format that eviction writes.

**Return type**
None

**Parameters**
**items** (*list[tuple[tuple[int, str], bytes]]*)

**class** angr.knowledge_plugins.structured_code.**StructuredCodeManager**

Bases: *KnowledgeBasePlugin*

A knowledge base plugin to store structured code generator results.

**__init__**(*kb*, *cache_limit=None*)

**Parameters**
**cache_limit** (*int | None*)

**get**(*key*, *default=None*)

**discard**(*key*)

**available_flavors**(*item*)

**all_flavors**(*item*)

**copy**()

### 10.22.18 angr.knowledge_plugins.types

**class** angr.knowledge_plugins.types.**TypesStore**

Bases: *KnowledgeBasePlugin*, UserDict[str, *TypeRef*]

A kb plugin that stores a mapping from name to TypeRef. It will return types from angr.sim_type.ALL_TYPES as a default.

**__init__**(*kb*)

**copy**()

**get_own**(*item*)

**Return type**
*TypeRef*

**Parameters**
**item** (*str*)

**iter_own**()

Iterate over all the names which are stored in this object - i.e. values() without ALL_TYPES

**iter_own_keys**()

**rename**(*old*, *new*)

**Parameters**

- **old** (*str*)
- **new** (*str*)

**unique_type_name**()

**Return type**
str

### 10.22.19 angr.knowledge_plugins.variables

**class** angr.knowledge_plugins.variables.**DecompilationVariableManager**

Bases: *VariableManager*

Holds variables discovered during decompilation, kept separate from the disassembly-level kb.variables. Exposed as kb.dec_variables. Per-function managers are held in a SpillingVariableInternalDict, which spills least-recently-used entries to the RuntimeDb LMDB store (disable via USE_SPILLING_DVARS).

**__init__**(*kb*)

**copy**()

**Return type**
*DecompilationVariableManager*

**class** angr.knowledge_plugins.variables.**VariableManager**

Bases: *KnowledgeBasePlugin*

Manage variables.

**__init__**(*kb*)

**function_managers**: dict[int, *VariableManagerInternal*] | *SpillingVariableInternalDict*

**has_function_manager**(*key*)

**Return type**
bool

**Parameters**
**key** (*int*)

**get_function_manager**(*func_addr*)

**Return type**
*VariableManagerInternal*

**initialize_variable_names**()

**Return type**
None

**get_variable_accesses**(*variable*, *same_name=False*)

Get a list of all references to the given variable.

**Parameters**

- **variable** (*SimVariable*) – The variable.
- **same_name** (bool) – Whether to include all variables with the same variable name, or just based on the variable identifier.

**Return type**
list[*VariableAccess*]

**Returns**
All references to the variable.

**copy**()

**static convert_variable_list**(*vlist*, *manager*)

**Parameters**

- **vlist** (*list[Variable]*)
- **manager** (VariableManagerInternal)

**load_from_dwarf**(*cu_list=None*)

**Parameters**
**cu_list** (*list[CompilationUnit] | None*)

**class** angr.knowledge_plugins.variables.**VariableType**

Bases: object

Describes variable types.

**REGISTER = 0**

**MEMORY = 1**

### Submodules


*spilling_vardict*
*variable_access*
*variable_manager*


### angr.knowledge_plugins.variables.spilling_vardict

**class** angr.knowledge_plugins.variables.spilling_vardict.**SpillingVariableInternalDict**

Bases: MutableMapping

A dict of per-function VariableManagerInternal instances (keyed by function address) that keeps only the most recently used cache_limit entries in memory and spills the rest to an LMDB database managed by the RuntimeDb knowledge base plugin. It is the backing store for `DecompilationVariableManager.function_managers`.

Evicted entries are serialized and written out; they are deserialized on access with the owning manager reattached. Decompilation only mutates a function's internal manager while decompiling that function (when it is the most-recently-used entry) and treats it as read-only afterwards, so spilling is safe.

On pickle, the serialized entries travel inside the pickle and are re-imported into a fresh LMDB on first access after unpickling.

**__init__**(*manager*, *cache_limit=1000*)

**Parameters**

- **manager** (DecompilationVariableManager)
- **cache_limit** (*int*)

**property cache_limit:** int

### angr.knowledge_plugins.variables.variable_access

**class** angr.knowledge_plugins.variables.variable_access.**VariableAccessSort**

Bases: object

Provides enums for variable access types.

**WRITE = 0**

**READ = 1**

**REFERENCE = 2**

**class** angr.knowledge_plugins.variables.variable_access.**VariableAccess**

Bases: *Serializable*

Describes a variable access.

**__init__**(*variable*, *access_type*, *location*, *offset*, *atom_hash=None*)

**variable:** *SimVariable*

**access_type:** int

**location:** *CodeLocation*

**offset:** int | None

**atom_hash:** int | None

### angr.knowledge_plugins.variables.variable_manager

**class** angr.knowledge_plugins.variables.variable_manager.**SortedDict**

Bases: dict[K, T], Generic

**irange**(**args*, ***kwargs*)

**Return type**
Iterator[TypeVar(K)]

**class** angr.knowledge_plugins.variables.variable_manager.**VariableType**

Bases: object

Describes variable types.

**REGISTER = 0**

**MEMORY = 1**

**class** angr.knowledge_plugins.variables.variable_manager.**LiveVariables**

Bases: object

A collection of live variables at a program point.

**__init__**(*register_region*, *stack_region*)

**register_region**

**stack_region**

**class** angr.knowledge_plugins.variables.variable_manager.**VariableManagerInternal**

Bases: *Serializable*

Manage variables for a function. It is meant to be used internally by VariableManager, but it's common to be given a reference to one in response to a query for "the variables for a given function". Maybe a better name would be "VariableManagerScope".

**__init__**(*manager*, *func_addr=None*)

**manager: *VariableManager***

**variable_to_types: dict[*SimVariable*, *SimType*]**

**array_length_exprs: dict[*SimVariable*, *Expression*]**

**set_manager**(*manager*)

**Parameters**
**manager** (VariableManager)

**next_variable_ident**(*sort*)

**add_variable**(*sort*, *start*, *variable*)

**Parameters**
**variable** (SimVariable)

**set_variable**(*sort*, *start*, *variable*)

**Parameters**
**variable** (SimVariable)

**write_to**(*variable*, *offset*, *location*, *overwrite=False*, *atom=None*)

**read_from**(*variable*, *offset*, *location*, *overwrite=False*, *atom=None*)

**reference_at**(*variable*, *offset*, *location*, *overwrite=False*, *atom=None*)

**record_variable**(*location*, *variable*, *offset*, *overwrite=False*, *atom=None*)

**Parameters**

- **location** (CodeLocation)
- **offset** (*int | None*)
- **atom** (Atom | *None*)

**remove_variable_by_atom**(*location*, *variable*, *atom*)

**Parameters**

- **location** (CodeLocation)
- **variable** (SimVariable)

**make_phi_node**(*block_addr*, **variables*)

Create a phi variable for variables at block *block_addr*.

**Parameters**

- **block_addr** (*int*) – The address of the current block.
- **variables** – Variables that the phi variable represents.

**Returns**

The created phi variable.

**set_live_variables**(*addr*, *register_region*, *stack_region*)

**find_variables_by_insn**(*ins_addr*, *sort*)

**is_variable_used_at**(*variable*, *loc*)

**Return type**

bool

**Parameters**

- **variable** (SimVariable)
- **loc** (*tuple[int, int]*)

**find_variable_by_stmt**(*block_addr*, *stmt_idx*, *sort*, *block_idx=None*)

**Parameters**

**block_idx** (*int | None*)

**find_variables_by_stmt**(*block_addr*, *stmt_idx*, *sort*, *block_idx=None*)

**Return type**

list[tuple[*SimVariable*, int | None]]

**Parameters**

- **block_addr** (*int*)
- **stmt_idx** (*int*)

- **sort** (*str*)
- **block_idx** (*int | None*)

**find_variable_by_atom**(*block_addr*, *stmt_idx*, *atom*, *block_idx=None*)

**Parameters**
**block_idx** (*int | None*)

**find_variables_by_atom**(*block_addr*, *stmt_idx*, *atom*, *block_idx=None*)

**Return type**
set[tuple[*SimVariable*, int | None]]

**Parameters**

- **atom** (Expression)
- **block_idx** (*int | None*)

**find_variables_by_stack_offset**(*offset*)

**Return type**
set[*SimVariable*]

**Parameters**
**offset** (*int*)

**find_variables_by_register**(*reg*)

**Return type**
set[*SimVariable*]

**Parameters**
**reg** (*str | int*)

**get_variable_accesses**(*variable*, *same_name=False*)

**Return type**
list[*VariableAccess*]

**Parameters**

- **variable** (SimVariable)
- **same_name** (*bool*)

**get_variables**(*sort=None*, *collapse_same_ident=False*)

Get a list of variables.

**Overloads**

- **self**, **sort** (Literal['stack']), **collapse_same_ident** (bool) → list[SimStackVariable]
- **self**, **sort** (Literal['reg']), **collapse_same_ident** (bool) → list[SimRegisterVariable]
- **self**, **sort** (None), **collapse_same_ident** (bool) → list[SimRegisterVariable]

**Parameters**

- **sort** – Sort of the variable to get.
- **collapse_same_ident** – Whether variables of the same identifier should be collapsed or not.

**Returns**
A list of variables.

`get_unified_variables`(*sort=None*)

Get a list of unified variables.

**Overloads**

- **self**, **sort** (Literal['stack']) → list[SimStackVariable]
- **self**, **sort** (Literal['reg']) → list[SimRegisterVariable]
- **self**, **sort** (None) → list[SimRegisterVariable]

**Parameters**
**sort** – Sort of the variable to get.

**Returns**
A list of variables.

`get_global_variables`(*addr*)

Get global variable by the address of the variable.

**Parameters**
**addr** (*int*) – Address of the variable.

**Returns**
A set of variables or an empty set if no variable exists.

`is_phi_variable`(*var*)

Test if *var* is a phi variable.

**Parameters**
**var** (SimVariable) – The variable instance.

**Returns**
True if *var* is a phi variable, False otherwise.

**Return type**
bool

`get_phi_subvariables`(*var*)

Get sub-variables that phi variable *var* represents.

**Parameters**
**var** (SimVariable) – The variable instance.

**Returns**
A set of sub-variables, or an empty set if *var* is not a phi variable.

**Return type**
set

`get_phi_variables`(*block_addr*)

Get a dict of phi variables and their corresponding variables.

**Parameters**
**block_addr** (*int*) – Address of the block.

**Returns**
A dict of phi variables of an empty dict if there are no phi variables at the block.

**Return type**
dict

**get_variables_without_writes**()

Get all variables that have never been written to.

**Return type**
list[*SimVariable*]

**Returns**
A list of variables that are never written to.

**input_variables**(*exclude_specials=True*)

Get all variables that have never been written to.

**Returns**
A list of variables that are never written to.

**Parameters**
**exclude_specials** (*bool*)

**assign_variable_names**(*labels=None*, *types=None*)

Assign default names to all SSA variables.

**Parameters**
**labels** – Known labels in the binary.

**Returns**
None

**assign_unified_variable_names**(*labels=None*, *arg_names=None*, *reset=False*, *func_blocks=None*)

Assign default names to all unified variables. If *func_blocks* is provided, we will find out variables that are only ever written to in Phi assignments and never used elsewhere, and put these variables at the end of the sorted list. These variables are likely completely removed during the dephication process.

**Parameters**

- **labels** – Known labels in the binary.
- **arg_names** (list[str] | None) – Known argument names.
- **reset** (bool) – Reset all variable names or not.
- **func_blocks** (list[Block] | None) – A list of function blocks of the function where these variables are accessed.

**Return type**
None

**set_variable_type**(*var*, *ty*, *name=None*, *override_bot=True*, *all_unified=False*, *mark_manual=False*)

**Return type**
None

**Parameters**

- **var** (SimVariable)
- **ty** (SimType)
- **name** (*str | None*)
- **override_bot** (*bool*)
- **all_unified** (*bool*)
- **mark_manual** (*bool*)

`get_variable_type`(*var*)

**Return type**
*SimType* | None

`remove_types`()

`unify_variables`(*interference=None*)

Map SSA variables to a unified variable. Fill in self._unified_variables.

**Parameters**
**interference** (*networkx.Graph[int] | None*)

**Return type**
None

`set_unified_variable`(*variable*, *unified*)

Set the unified variable for a given SSA variable.

**Parameters**

- **variable** (*SimVariable*) – The SSA variable.
- **unified** (*SimVariable*) – The unified variable.

**Return type**
None

**Returns**
None

`unified_variable`(*variable*)

Return the unified variable for a given SSA variable,

**Parameters**
**variable** (*SimVariable*) – The SSA variable.

**Return type**
*SimVariable* | None

**Returns**
The unified variable, or None if there is no such SSA variable.

`get_stackvar_max_sizes`(*stack_items*)

Get the maximum size of each stack variable regardless of the type of each stack variable, under the assumption that stack variables do not overlap.

**Return type**
dict[*SimStackVariable*, int]

**Returns**
A dictionary from SimStackVariable to its maximum size.

**Parameters**
**stack_items** (*dict[int, StackItem]*)

**class** angr.knowledge_plugins.variables.variable_manager.**VariableManager**

Bases: *KnowledgeBasePlugin*

Manage variables.

`__init__`(*kb*)

**function_managers**: dict[int, *VariableManagerInternal*] | *SpillingVariableInternalDict*

**has_function_manager**(*key*)

**Return type**
bool

**Parameters**
**key** (*int*)

**get_function_manager**(*func_addr*)

**Return type**
*VariableManagerInternal*

**initialize_variable_names**()

**Return type**
None

**get_variable_accesses**(*variable*, *same_name=False*)

Get a list of all references to the given variable.

**Parameters**

- **variable** (*SimVariable*) – The variable.
- **same_name** (bool) – Whether to include all variables with the same variable name, or just based on the variable identifier.

**Return type**
list[*VariableAccess*]

**Returns**
All references to the variable.

**copy**()

**static convert_variable_list**(*vlist*, *manager*)

**Parameters**

- **vlist** (*list[Variable]*)
- **manager** (VariableManagerInternal)

**load_from_dwarf**(*cu_list=None*)

**Parameters**
**cu_list** (*list[CompilationUnit] | None*)

**class** angr.knowledge_plugins.variables.variable_manager.**DecompilationVariableManager**

Bases: *VariableManager*

Holds variables discovered during decompilation, kept separate from the disassembly-level kb.variables. Exposed as kb.dec_variables. Per-function managers are held in a SpillingVariableInternalDict, which spills least-recently-used entries to the RuntimeDb LMDB store (disable via USE_SPILLING_DVARS).

**__init__**(*kb*)

copy()

**Return type**
*DecompilationVariableManager*

### 10.22.20 angr.knowledge_plugins.xrefs

class angr.knowledge_plugins.xrefs.**XRef**

Bases: *Serializable*

XRef describes a reference to a MemoryData instance (if a MemoryData instance is available) or just an address.

__init__(*ins_addr=None*, *block_addr=None*, *stmt_idx=None*, *insn_op_idx=None*, *memory_data=None*, *dst=None*, *xref_type=None*)

**Parameters**

- **ins_addr** (*int | None*)
- **block_addr** (*int | None*)
- **stmt_idx** (*int | None*)
- **insn_op_idx** (*int | None*)
- **memory_data** (*MemoryData | None*)
- **dst** (*int | None*)

**ins_addr:** int | None

**insn_op_idx:** int | None

**block_addr:** int | None

**stmt_idx:** int | None

**memory_data**

**type**

**dst**

**property type_string**

**insn_op_type**

**copy**()

class angr.knowledge_plugins.xrefs.**XRefManager**

Bases: *KnowledgeBasePlugin*, *Serializable*

__init__(*kb*)

**copy**()

**clear**()

**add_xref**(*xref*)

**add_xrefs**(*xrefs*)

**get_xrefs_by_ins_addr**(*ins_addr*)

**get_xrefs_by_dst**(*dst*)

**get_xrefs_by_dst_region**(*start*, *end*)

Get a set of XRef objects that point to a given address region bounded by start and end. Will only return absolute xrefs, not relative ones (like SP offsets)

**get_next_xref_addr_by_dst**(*addr*)

Get the next XRef object whose address is greater than or equal to the given address. Will only return absolute xrefs, not relative ones (like SP offsets)

**Return type**
int | None

**Parameters**
**addr** (*int*)

**get_xrefs_by_ins_addr_region**(*start*, *end*)

Get a set of XRef objects that originate at a given address region bounded by start and end. Useful for finding references from a basic block or function.

**Return type**
set[*XRef*]

**class** angr.knowledge_plugins.xrefs.**XRefType**

Bases: object

**Offset = 0**

**Read = 1**

**Write = 2**

**static to_string**(*ty*)

## Submodules


*xref*
*xref_manager*
*xref_types*


### angr.knowledge_plugins.xrefs.xref

**class** angr.knowledge_plugins.xrefs.xref.**XRef**

Bases: *Serializable*

XRef describes a reference to a MemoryData instance (if a MemoryData instance is available) or just an address.

**__init__**(*ins_addr=None*, *block_addr=None*, *stmt_idx=None*, *insn_op_idx=None*, *memory_data=None*, *dst=None*, *xref_type=None*)

**Parameters**

- **ins_addr** (*int | None*)
- **block_addr** (*int | None*)

- **stmt_idx** (*int* | *None*)
- **insn_op_idx** (*int* | *None*)
- **memory_data** (MemoryData | *None*)
- **dst** (*int* | *None*)

**ins_addr:** int | None

**insn_op_idx:** int | None

**block_addr:** int | None

**stmt_idx:** int | None

**memory_data**

**type**

**dst**

**property type_string**

**insn_op_type**

**copy**()

### angr.knowledge_plugins.xrefs.xref_manager

**class** angr.knowledge_plugins.xrefs.xref_manager.**XrefDict**

Bases: SortedDict

A SortedDict that maps addresses to sets of XRefs. It adds defaultdict-like behavior around SortedDict.

**get_xrefs_in_range**(*start*, *end*)

Get a set of XRef objects that point to addresses in the given range.

**class** angr.knowledge_plugins.xrefs.xref_manager.**XRefManager**

Bases: *KnowledgeBasePlugin*, *Serializable*

**__init__**(*kb*)

**copy**()

**clear**()

**add_xref**(*xref*)

**add_xrefs**(*xrefs*)

**get_xrefs_by_ins_addr**(*ins_addr*)

**get_xrefs_by_dst**(*dst*)

**get_xrefs_by_dst_region**(*start*, *end*)

Get a set of XRef objects that point to a given address region bounded by start and end. Will only return absolute xrefs, not relative ones (like SP offsets)

**get_next_xref_addr_by_dst**(*addr*)

Get the next XRef object whose address is greater than or equal to the given address. Will only return absolute xrefs, not relative ones (like SP offsets)

**Return type**
int | None

**Parameters**
**addr** (*int*)

**get_xrefs_by_ins_addr_region**(*start*, *end*)

Get a set of XRef objects that originate at a given address region bounded by start and end. Useful for finding references from a basic block or function.

**Return type**
set[*XRef*]

### angr.knowledge_plugins.xrefs.xref_types

**class** angr.knowledge_plugins.xrefs.xref_types.**XRefType**

Bases: object

**Offset = 0**

**Read = 1**

**Write = 2**

**static to_string**(*ty*)

## 10.23 angr.llm_client

**class** angr.llm_client.**AgentKind**

Bases: Enum

The kind of pydantic-ai Agent to retrieve from the LLMClient cache.

**STRING = 'string'**

**STRUCTURED = 'structured'**

**class** angr.llm_client.**LLMClient**

Bases: object

A client for interacting with LLMs via pydantic-ai. Used by the decompiler to suggest improved variable names, function names, and types.

**__init__**(*model*, *api_key=None*, *api_base=None*, *max_tokens=4096*, *temperature=0.0*)

**Parameters**

- **model** (*str*)
- **api_key** (*str | None*)
- **api_base** (*str | None*)
- **max_tokens** (*int*)
- **temperature** (*float*)

**infer_provider**(*provider*)

Infer the provider from the provider name.

**Return type**
Provider[Any]

**Parameters**
**provider** (*str*)

**classmethod from_env**()

Create an LLMClient from environment variables.

Uses ANGR_LLM_MODEL (required), ANGR_LLM_API_KEY (optional), and ANGR_LLM_API_BASE (optional). Returns None if ANGR_LLM_MODEL is not set.

**Return type**
*LLMClient* | None

**completion**(*messages*, ***kwargs*)

Call the LLM with the given messages and return the response text.

**Return type**
str

**Parameters**
**messages** (*list[dict[str, str]]*)

**completion_structured**(*messages*, *output_type*, *raise_exc=False*, ***kwargs*)

Call the LLM with the given messages and return a validated Pydantic model. Returns None if the call fails.

**Parameters**

- **raise_exc** (bool) – If True, exceptions are propagated to the caller instead of being caught.
- **messages** (*list[dict[str, str]]*)
- **output_type** (*type[T]*)

**Return type**
Optional[TypeVar(T)]

**completion_json**(*messages*, *raise_exc=False*, ***kwargs*)

Call the LLM and parse the response as JSON. Strips markdown code fences if present. Returns None on parse failure. Kept for backwards compatibility; prefer completion_structured().

**Parameters**

- **raise_exc** (bool) – If True, exceptions are propagated to the caller instead of being caught.
- **messages** (*list[dict[str, str]]*)

**Return type**
dict | None

## 10.24 angr.llm_models

**class** angr.llm_models.**VariableRename**

Bases: BaseModel

**old_name:** str

**new_name:** str

**model_config:** ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class angr.llm_models.**VariableNameSuggestions**

Bases: BaseModel

**renames:** list[*VariableRename*]

**model_config:** ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class angr.llm_models.**FunctionNameSuggestion**

Bases: BaseModel

**function_name:** str

**model_config:** ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class angr.llm_models.**VariableTypeChange**

Bases: BaseModel

**variable_name:** str

**new_type:** str

**model_config:** ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

class angr.llm_models.**VariableTypeSuggestions**

Bases: BaseModel

**type_changes:** list[*VariableTypeChange*]

**model_config:** ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

## 10.25 angr.mcp

angr MCP Server - Model Context Protocol interface for angr binary analysis.

This module provides an MCP server that exposes angr's binary analysis capabilities through standardized tools accessible by LLM applications.

**Usage:**

# Run as module with stdio transport (for Claude Desktop, etc.) python -m angr.mcp

# Run with SSE transport python -m angr.mcp –transport sse –port 8080

# Run with HTTP transport python -m angr.mcp –transport http –host 0.0.0.0 –port 8080 –path /mcp

# Programmatic usage from angr.mcp import create_server server = create_server() server.run()

**Available Tools:**

- load_binary: Load a binary file for analysis
- get_cfg: Build the control flow graph
- recover_calling_conventions: Recover calling conventions and prototypes
- list_functions: List discovered functions
- get_function_info: Get detailed function information
- decompile_function: Decompile to pseudocode
- get_xrefs: Get cross-references
- get_strings: Extract strings from binary
- get_imports: List imported symbols
- get_exports: List exported symbols
- get_basic_blocks: Get basic blocks with disassembly
- get_callgraph: Get function call graph
- find_functions_by_pattern: Search functions by name
- list_projects: List active analysis sessions
- close_project: Close a project session

**Editing Tools (batch; see angr.mcp.edit_tools):**

- rename: Rename functions, decompilation variables, and globals
- set_type: Set variable types, globals, return types, and function prototypes
- set_comments: Set or clear comments at addresses

**exception** angr.mcp.**CFGNotBuiltError**

Bases: *MCPAngrError*

Raised when CFG is required but not built.

**exception** angr.mcp.**DecompilationError**

Bases: *MCPAngrError*

Raised when decompilation fails.

**exception** angr.mcp.**FunctionNotFoundError**

Bases: *MCPAngrError*

Raised when a function cannot be found.

**exception** angr.mcp.**MCPAngrError**

Bases: *AngrError*, ToolError

Base exception for MCP angr server errors.

Also a fastmcp ToolError: FastMCP masks the message of anything else, so a client would otherwise see a generic failure instead of the reason.

**exception** angr.mcp.**ProjectNotFoundError**

Bases: *MCPAngrError*

Raised when a project ID is not found.

**class** angr.mcp.**ProjectSession**

Bases: object

Represents a loaded angr project with its associated analyses.

**project_id:** *str*

**project:** *Project*

**binary_path:** *str*

**cfg:** *CFGModel* | None = None

**lock:** *RLock*

**property has_cfg:** *bool*

Check if CFG has been built for this project.

**exclusive**()

Serialize access to this project.

FastMCP runs synchronous tools on worker threads, so two calls can reach the same Project and KnowledgeBase at once. Neither is thread-safe, and the decompilation and variable caches spill to LMDB, where a concurrent eviction and reload can hand back a half-imported object. Read-only tools are not exempt: a read racing an eviction is exactly that hazard.

**Return type**
Iterator[*ProjectSession*]

**__init__**(*project_id*, *project*, *binary_path*, *cfg=None*, *lock=<factory>*)

**Parameters**

- **project_id** (*str*)
- **project** (Project)
- **binary_path** (*str*)
- **cfg** (CFGModel | *None*)
- **lock** (*RLock*)

**Return type**
None

**class** angr.mcp.**SessionManager**

Bases: object

Manages multiple angr project sessions.

This class maintains state across MCP tool invocations, allowing projects to persist between calls.

**__init__**()

**Return type**
None

**create_session**(*binary_path*, ***kwargs*)

Load a binary and create a new project session.

**Parameters**

- **binary_path** (str) – Path to the binary to analyze
- **kwargs** (Any) – Additional arguments passed to angr.Project

**Return type**
*ProjectSession*

**Returns**
The created ProjectSession with unique ID

**get_session**(*project_id*)

Get an existing session by ID.

**Parameters**
**project_id** (str) – The project ID to look up

**Return type**
*ProjectSession*

**Returns**
The ProjectSession

**Raises**
**KeyError** – If no project with that ID exists

**list_sessions**()

List all active sessions.

**Return type**
list[dict[str, Any]]

**Returns**
List of session metadata dictionaries

**close_session**(*project_id*)

Close and remove a session.

**Parameters**
**project_id** (str) – The project ID to close

**Return type**
bool

**Returns**
True if closed, False if not found

angr.mcp.**create_server**()

Create and return the configured MCP server instance.

**Return type**
FastMCP

angr.mcp.**get_session_manager**()

Get or create the global session manager.

**Return type**
*SessionManager*

### Submodules

| | |
|---|---|
| *edit_tools* | Batch edit tools: rename, set_type, and set_comments. |
| *errors* | |
| *serializers* | |
| *server* | Main MCP server for angr binary analysis. |
| *session* | Session management for angr MCP server. |

## 10.25.1 angr.mcp.edit_tools

Batch edit tools: rename, set_type, and set_comments.

Every tool takes a list of items and returns one result per item. Per-item failures are reported as data rather than raised, so one bad item does not discard the work done by its siblings – that is the point of batching. Only whole-call problems (an unknown project, a malformed request) raise.

angr.mcp.edit_tools.**rename**(*project_id*, *items*, *dry_run=False*, *stop_on_error=False*, *allow_overwrite=False*, *auto_decompile=True*)

Rename functions, decompilation variables, and globals in one call.

Each item needs a "kind" and a "new_name":

- kind "function": identify it with "name" or "address".
- kind "variable": "function" (name or address) plus "name", the variable's current name as it appears in the pseudocode. Covers locals, arguments, and stack slots – angr routes them through one path, so the item does not say which; the result reports the resolved storage.
- kind "global": "address", or "name" if the global already has a label.

Renames persist across re-decompilation. Renaming a variable requires the function to have been decompiled; with auto_decompile it is decompiled on demand.

**Parameters**

- **project_id** (str) – The project ID
- **items** (list[dict[str, Any]]) – The rename requests
- **dry_run** (bool) – Resolve and validate without changing anything. Not side-effect free: resolving a variable name may decompile the function.
- **stop_on_error** (bool) – Stop at the first failure and mark the rest skipped. Edits already applied are NOT rolled back; use dry_run as a pre-flight.
- **allow_overwrite** (bool) – Permit a name already bound to something else (default: False)
- **auto_decompile** (bool) – Decompile a function on demand when a variable rename needs it

**Return type**
dict[str, Any]

**Returns**
A per-item result list plus counts of succeeded/unchanged/failed/skipped

angr.mcp.edit_tools.**set_type**(*project_id*, *items*, *dry_run=False*, *stop_on_error=False*, *redecompile=False*, *auto_decompile=True*)

Set types on decompilation variables, globals, and function prototypes in one call.

Each item needs a "kind" and a "type":

- kind “variable”: “function” plus “name”. Retyping an argument rewrites the function’s prototype, which forces that function to be re-decompiled.
- kind “function”: identify it with “function”, “name”, or “address”; “type” is a full signature, e.g. “int parse(char *buf, int len)”. The name inside the signature is ignored – use rename for that. This discards the function’s cached decompilation and variables; earlier renames and manual types, including ones set by earlier items in the same batch, are restored afterwards.
- kind “return”: same identifiers, replaces only the return type and keeps the arguments.
- kind “global”: “address” or “name”, plus “type”.

Types are re-inferred through the cached constraints once per affected function rather than per item. That reflow does NOT rebuild the syntax tree, so a retype that should change how an access renders – a struct field, an array index – needs redecompile=True.

**Parameters**

- **project_id** (str) – The project ID
- **items** (list[dict[str, Any]]) – The type requests
- **dry_run** (bool) – Validate and resolve without changing anything
- **stop_on_error** (bool) – Stop at the first failure; already-applied edits are NOT rolled back
- **redecompile** (bool) – Fully re-decompile each affected function instead of reflowing types
- **auto_decompile** (bool) – Decompile a function on demand when an item needs it

**Return type**
dict[str, Any]

**Returns**
A per-item result list plus counts of succeeded/unchanged/failed/skipped

angr.mcp.edit_tools.**set_comments**(*project_id*, *items*, *dry_run=False*, *stop_on_error=False*)

Set or clear comments at addresses in one call.

Each item needs a “kind” and a “comment”; an empty comment clears it.

- kind “address”: “address” is any address in the binary.
- kind “function”: identify it with “function”, “name”, or “address”; comments its header.

A comment is written to the knowledge base – where the disassembly and the function header read it – and mirrored next to the matching pseudocode statement. Because pseudocode comments are keyed by instruction address, an address that is not a statement boundary is snapped down to the nearest one; each result reports “snapped_from” and whether the comment actually rendered inline (“rendered_inline”) rather than in the orphaned-comments block.

**Parameters**

- **project_id** (str) – The project ID
- **items** (list[dict[str, Any]]) – The comment requests
- **dry_run** (bool) – Resolve addresses without changing anything
- **stop_on_error** (bool) – Stop at the first failure; already-applied edits are NOT rolled back

**Return type**
dict[str, Any]

**Returns**
A per-item result list plus counts of succeeded/unchanged/failed/skipped

### 10.25.2 angr.mcp.errors

**exception** angr.mcp.errors.**MCPAngrError**

Bases: *AngrError*, ToolError

Base exception for MCP angr server errors.

Also a fastmcp ToolError: FastMCP masks the message of anything else, so a client would otherwise see a generic failure instead of the reason.

**exception** angr.mcp.errors.**ProjectNotFoundError**

Bases: *MCPAngrError*

Raised when a project ID is not found.

**exception** angr.mcp.errors.**CFGNotBuiltError**

Bases: *MCPAngrError*

Raised when CFG is required but not built.

**exception** angr.mcp.errors.**FunctionNotFoundError**

Bases: *MCPAngrError*

Raised when a function cannot be found.

**exception** angr.mcp.errors.**DecompilationError**

Bases: *MCPAngrError*

Raised when decompilation fails.

**exception** angr.mcp.errors.**DecompilationComplexityError**

Bases: *DecompilationError*

Raised when a function is too large for the decompiler's complexity limits.

The message names the limit and the measured size so the caller can decide whether to raise the limit or skip the function.

**exception** angr.mcp.errors.**InvalidArgumentError**

Bases: *MCPAngrError*, ValueError

Raised when a tool is called with unusable arguments.

### 10.25.3 angr.mcp.serializers

angr.mcp.serializers.**serialize_function**(*func*, *include_blocks=False*)

Serialize a Function object to a JSON-compatible dict.

**Parameters**

- **func** (*Function*) – The angr Function to serialize
- **include_blocks** (bool) – Whether to include block addresses

**Return type**

dict[str, Any]

**Returns**

Dictionary with function information

angr.mcp.serializers.**serialize_function_summary**(*func*)

Serialize minimal function info for list views.

**Parameters**
**func** (*Function*) – The angr Function to serialize

**Return type**
dict[str, Any]

**Returns**
Dictionary with minimal function information

angr.mcp.serializers.**serialize_xref**(*xref*)

Serialize an XRef object.

**Parameters**
**xref** (*XRef*) – The angr XRef to serialize

**Return type**
dict[str, Any]

**Returns**
Dictionary with cross-reference information

angr.mcp.serializers.**serialize_basic_block**(*block*, *include_disasm=True*)

Serialize a basic block.

**Parameters**

- **block** (Any) – The angr Block to serialize
- **include_disasm** (bool) – Whether to include disassembly

**Return type**
dict[str, Any]

**Returns**
Dictionary with block information

angr.mcp.serializers.**serialize_cfg_stats**(*cfg_model*)

Serialize CFG statistics.

**Parameters**
**cfg_model** (*CFGModel*) – The angr CFGModel to summarize

**Return type**
dict[str, Any]

**Returns**
Dictionary with CFG statistics

angr.mcp.serializers.**serialize_symbol**(*symbol*)

Serialize a CLE symbol.

**Parameters**
**symbol** (Any) – The CLE Symbol to serialize

**Return type**
dict[str, Any]

**Returns**
Dictionary with symbol information

### 10.25.4 angr.mcp.server

Main MCP server for angr binary analysis.

**class** angr.mcp.server.**StdoutGuardMiddleware**

Bases: Middleware

Keep analysis output off of stdout.

Under the stdio transport, stdout is the JSON-RPC channel. This middleware redirects stdout to stderr to avoid messing with the stdio transport.

**async on_message**(*context*, *call_next*)

**Return type**
Any

**Parameters**

- **context** (*MiddlewareContext[Any]*)
- **call_next** (*CallNext[Any, Any]*)

angr.mcp.server.**load_binary**(*binary_path*, *auto_load_libs=False*)

Load a binary file as an angr Project for analysis.

This is typically the first tool to call. It creates a new analysis session and returns a project_id that must be used for subsequent operations.

**Parameters**

- **binary_path** (str) – Absolute path to the binary file to analyze
- **auto_load_libs** (bool) – Whether to automatically load shared libraries (default: False)

**Return type**
dict[str, Any]

**Returns**
Project information including project_id, architecture, and entry point

angr.mcp.server.**get_cfg**(*project_id*, *normalize=True*, *data_references=True*)

Build or retrieve the Control Flow Graph (CFG) for a project.

The CFG is required for most analysis operations including function discovery, decompilation, and cross-reference analysis.

**Parameters**

- **project_id** (str) – The project ID returned by load_binary
- **normalize** (bool) – Whether to normalize the CFG (recommended: True)
- **data_references** (bool) – Whether to collect data references (for string analysis)

**Return type**
dict[str, Any]

**Returns**
CFG statistics including node/edge counts and functions discovered

angr.mcp.server.**recover_calling_conventions**(*project_id*, *workers=0*, *force=False*)

Recover calling conventions and prototypes for every function in the binary.

Requires CFG to be built first via get_cfg. This improves the quality of decompilation output, especially the arguments shown at call sites.

**Parameters**

- **project_id** (str) – The project ID
- **workers** (int) – Number of worker processes to use (0 = run in-process, the default)
- **force** (bool) – Re-analyze functions that already have a calling convention or prototype

**Return type**
dict[str, Any]

**Returns**
Number of functions for which a calling convention was recovered

angr.mcp.server.**list_functions**(*project_id*, *filter_plt=None*, *filter_syscall=None*, *name_pattern=None*, *limit=100*, *offset=0*)

List all functions discovered in the binary.

Requires CFG to be built first via get_cfg.

**Parameters**

- **project_id** (str) – The project ID
- **filter_plt** (bool | None) – If True, only show PLT stubs; if False, exclude them
- **filter_syscall** (bool | None) – If True, only show syscalls; if False, exclude them
- **name_pattern** (str | None) – Filter functions by name substring (case-insensitive)
- **limit** (int) – Maximum number of functions to return (default: 100)
- **offset** (int) – Number of functions to skip (for pagination)

**Return type**
dict[str, Any]

**Returns**
List of function summaries with addresses and names

angr.mcp.server.**get_function_info**(*project_id*, *address=None*, *name=None*, *include_blocks=False*)

Get detailed information about a specific function.

Specify either address (hex string like "0x401000") or function name.

**Parameters**

- **project_id** (str) – The project ID
- **address** (str | None) – Function address as hex string (e.g., "0x401000")
- **name** (str | None) – Function name to look up
- **include_blocks** (bool) – Whether to include list of basic block addresses

**Return type**
dict[str, Any]

**Returns**
Detailed function information including size, complexity, etc. "num_blocks" against "decompilation_block_limit" tells you up front whether decompile_function will refuse this function as too complex.

angr.mcp.server.**decompile_function**(*project_id*, *address=None*, *name=None*, *max_function_blocks=None*, *max_ail_statements=None*)

Decompile a function to C-like pseudocode.

Specify either address (hex string) or function name.

Pathologically large functions (e.g. packer filler that linear disassembly turns into thousands of blocks) can keep the decompiler busy indefinitely, so two size limits are checked before the expensive work starts. Exceeding either one fails with an error naming the limit and the measured size, so you can retry with a larger limit or move on to another function.

**Parameters**

- **project_id** (str) – The project ID
- **address** (str | None) – Function address as hex string (e.g., "0x401000")
- **name** (str | None) – Function name to decompile
- **max_function_blocks** (int | None) – Refuse to decompile a function whose CFG has more basic blocks than this (default: 50000; 0 disables the check). get_function_info reports a function's block count as "num_blocks" and this default as "decompilation_block_limit".
- **max_ail_statements** (int | None) – Refuse to decompile a function whose AIL graph has more statements than this (default: 1000000; 0 disables the check)

**Return type**
dict[str, Any]

**Returns**
Decompiled pseudocode and function metadata

angr.mcp.server.**get_xrefs**(*project_id*, *address*, *direction='to'*)

Get cross-references to or from a specific address.

**Parameters**

- **project_id** (str) – The project ID
- **address** (str) – The address to query (hex string, e.g., "0x401000")
- **direction** (str) – "to" for references TO this address, "from" for references FROM this address

**Return type**
dict[str, Any]

**Returns**
List of cross-references with source/destination and type

angr.mcp.server.**get_strings**(*project_id*, *min_length=4*, *limit=100*)

Extract strings from the binary.

Requires CFG with data_references=True.

**Parameters**

- **project_id** (str) – The project ID
- **min_length** (int) – Minimum string length to include (default: 4)
- **limit** (int) – Maximum number of strings to return (default: 100)

**Return type**
dict[str, Any]

**Returns**
List of strings with their addresses

angr.mcp.server.**get_imports**(*project_id*)

List imported symbols (external functions/variables the binary depends on).

**Parameters**
**project_id** (str) – The project ID

**Return type**
dict[str, Any]

**Returns**
List of imported symbols with their names and addresses

angr.mcp.server.**get_exports**(*project_id*)

List exported symbols (functions/variables this binary provides).

**Parameters**
**project_id** (str) – The project ID

**Return type**
dict[str, Any]

**Returns**
List of exported symbols with their names and addresses

angr.mcp.server.**get_basic_blocks**(*project_id*, *function_address*, *include_disasm=True*)

Get all basic blocks for a function.

**Parameters**

- **project_id** (str) – The project ID
- **function_address** (str) – Function address as hex string
- **include_disasm** (bool) – Whether to include disassembly for each block

**Return type**
dict[str, Any]

**Returns**
List of basic blocks with their instructions

angr.mcp.server.**get_callgraph**(*project_id*, *max_depth=None*, *root_address=None*)

Get the function call graph.

**Parameters**

- **project_id** (str) – The project ID
- **max_depth** (int | None) – Maximum depth from root (None for full graph)
- **root_address** (str | None) – Optional starting function address for subgraph

**Return type**
dict[str, Any]

**Returns**
Call graph as nodes and edges

angr.mcp.server.**find_functions_by_pattern**(*project_id*, *pattern*, *search_type='contains'*)

Search for functions by name pattern.

**Parameters**

- **project_id** (str) – The project ID
- **pattern** (str) – Search pattern (depends on search_type)
- **search_type** (str) – “contains”, “startswith”, “endswith”, or “regex”

**Return type**
dict[str, Any]

**Returns**
List of matching functions

angr.mcp.server.**list_projects**()

List all currently loaded projects/sessions.

**Return type**
dict[str, Any]

**Returns**
List of active project sessions with their IDs and metadata

angr.mcp.server.**close_project**(*project_id*)

Close a project and free its resources.

**Parameters**
**project_id** (str) – The project ID to close

**Return type**
dict[str, Any]

**Returns**
Confirmation of closure

angr.mcp.server.**create_server**()

Create and return the configured MCP server instance.

**Return type**
FastMCP

### 10.25.5 angr.mcp.session

Session management for angr MCP server.

**class** angr.mcp.session.**ProjectSession**

Bases: object

Represents a loaded angr project with its associated analyses.

**project_id:** str

**project:** *Project*

**binary_path:** str

**cfg:** *CFGModel* | None = None

**lock:** RLock

**property has_cfg:** bool

Check if CFG has been built for this project.

**exclusive**()

Serialize access to this project.

FastMCP runs synchronous tools on worker threads, so two calls can reach the same Project and KnowledgeBase at once. Neither is thread-safe, and the decompilation and variable caches spill to LMDB, where a concurrent eviction and reload can hand back a half-imported object. Read-only tools are not exempt: a read racing an eviction is exactly that hazard.

**Return type**
Iterator[*ProjectSession*]

**__init__**(*project_id*, *project*, *binary_path*, *cfg=None*, *lock=<factory>*)

**Parameters**

- **project_id** (*str*)
- **project** (Project)
- **binary_path** (*str*)
- **cfg** (CFGModel | *None*)
- **lock** (*RLock*)

**Return type**
None

**class** angr.mcp.session.**SessionManager**

Bases: object

Manages multiple angr project sessions.

This class maintains state across MCP tool invocations, allowing projects to persist between calls.

**__init__**()

**Return type**
None

**create_session**(*binary_path*, ***kwargs*)

Load a binary and create a new project session.

**Parameters**

- **binary_path** (*str*) – Path to the binary to analyze
- **kwargs** (Any) – Additional arguments passed to angr.Project

**Return type**
*ProjectSession*

**Returns**
The created ProjectSession with unique ID

**get_session**(*project_id*)

Get an existing session by ID.

**Parameters**
**project_id** (*str*) – The project ID to look up

**Return type**
*ProjectSession*

**Returns**
The ProjectSession

**Raises**
**KeyError** – If no project with that ID exists

**list_sessions**()

List all active sessions.

**Return type**
list[dict[str, Any]]

**Returns**
List of session metadata dictionaries

**close_session**(*project_id*)

Close and remove a session.

**Parameters**
**project_id** (str) – The project ID to close

**Return type**
bool

**Returns**
True if closed, False if not found

angr.mcp.session.**get_session_manager**()

Get or create the global session manager.

**Return type**
*SessionManager*

## 10.26 angr.misc

**class** angr.misc.**HookSet**

Bases: object

A HookSet is a static class that provides the capability to apply many hooks to an object.

**static install_hooks**(*target*, ***hooks*)

Given the target *target*, apply the hooks given as keyword arguments to it. If any targeted method has already been hooked, the hooks will not be overridden but will instead be pushed into a list of pending hooks. The final behavior should be that all hooks call each other in a nested stack.

**Parameters**

- **target** – Any object. Its methods named as keys in *hooks* will be replaced by *HookedMethod* objects.
- **hooks** – Any keywords will be interpreted as hooks to apply. Each method named will hooked with the corresponding function value.

**static remove_hooks**(*target*, ***hooks*)

Remove the given hooks from the given target.

**Parameters**

- **target** – The object from which to remove hooks. If all hooks are removed from a given method, the HookedMethod object will be removed and replaced with the original function.

- **hooks** – Any keywords will be interpreted as hooks to remove. You must provide the exact hook that was applied so that it can it can be identified for removal among any other hooks.

**static copy_hooks**(*source*, *target*, *domain*)

Copy the hooks from source onto target.

If the current callstack includes hooked methods from source, the already-called methods will not be included in the copy.

domain is a list of names that might be hooked.

**class** angr.misc.**Loggers**

Bases: object

Implements a loggers manager for angr.

**__init__**(*default_level=30*)

**default_level**

**profiling_enabled**

**handler**

**IN_SCOPE = ('angr', 'claripy', 'cle', 'pyvex', 'archinfo', 'tracer', 'driller', 'rex', 'patcherex', 'identifier')**

**load_all_loggers**()

A dumb and simple way to conveniently aggregate all loggers.

Adds attributes to this instance of each registered logger, replacing '.' with '_'

**enable_root_logger**()

Enable angr's default logger

**disable_root_logger**()

Disable angr's default logger

**static setall**(*level*)

**class** angr.misc.**PicklableLock**

Bases: object

Normal thread-locks are not pickleable. This provides a pickleable lock by mandating that the lock is unlocked during serialization.

**__init__**(**args*, ***kwargs*)

**acquire**(**args*, ***kwargs*)

**locked**()

**release**()

**class** angr.misc.**PluginHub**

Bases: Generic

A plugin hub is an object which contains many plugins, as well as the notion of a "preset", or a backer that can provide default implementations of plugins which cater to a certain circumstance.

Objects in angr like the SimState, the Analyses hub, the SimEngine selector, etc all use this model to unify their mechanisms for automatically collecting and selecting components to use. If you're familiar with design patterns this is a configurable Strategy Pattern.

Each PluginHub subclass should have a corresponding Plugin subclass, and perhaps a PluginPreset subclass if it wants its presets to be able to specify anything more interesting than a list of defaults.

**__init__**()

**Return type**
None

**classmethod register_default**(*name*, *plugin_cls*, *preset='default'*)

**Return type**
None

**Parameters**

- **name** (*str*)
- **plugin_cls** (type)
- **preset** (*str*)

**classmethod register_preset**(*name*, *preset*)

Register a preset instance with the class of the hub it corresponds to. This allows individual plugin objects to automatically register themselves with a preset by using a classmethod of their own with only the name of the preset to register with.

**Return type**
None

**Parameters**

- **name** (*str*)
- **preset** (PluginPreset*[P]*)

**property plugin_preset: *PluginPreset*[P] | None**

Get the current active plugin preset

**property has_plugin_preset: bool**

Check whether or not there is a plugin preset in use on this hub right now

**use_plugin_preset**(*preset*)

Apply a preset to the hub. If there was a previously active preset, discard it.

Preset can be either the string name of a preset or a PluginPreset instance.

**Return type**
None

**Parameters**
**preset** (*str* | PluginPreset*[P]*)

**discard_plugin_preset**()

Discard the current active preset. Will release any active plugins that could have come from the old preset.

**Return type**
None

**get_plugin**(*name*)

Get the plugin named name. If no such plugin is currently active, try to activate a new one using the current preset.

**Return type**
TypeVar(P)

**Parameters**
**name** (*str*)

**has_plugin**(*name*)

Return whether or not a plugin with the name name is currently active.

**Return type**
bool

**Parameters**
**name** (*str*)

**register_plugin**(*name*, *plugin*)

Add a new plugin plugin with name name to the active plugins.

**Return type**
TypeVar(P)

**Parameters**

- **name** (*str*)
- **plugin** (*P*)

**release_plugin**(*name*)

Deactivate and remove the plugin with name name.

**Return type**
None

**Parameters**
**name** (*str*)

**class** angr.misc.**PluginPreset**

Bases: Generic

A plugin preset object contains a mapping from name to a plugin class. A preset can be active on a hub, which will cause it to handle requests for plugins which are not already present on the hub.

Unlike Plugins and PluginHubs, instances of PluginPresets are defined on the module level for individual presets. You should register the preset instance with a hub to allow plugins to easily add themselves to the preset without an explicit reference to the preset itself.

**__init__**()

**Return type**
None

**activate**(*hub*)

This method is called when the preset becomes active on a hub.

**Return type**
None

**Parameters**
**hub** (PluginHub[P])

**deactivate**(*hub*)

This method is called when the preset is discarded from the hub.

**Return type**
None

**Parameters**
**hub** (PluginHub[P])

**add_default_plugin**(*name*, *plugin_cls*)

Add a plugin to the preset.

**Return type**
None

**Parameters**

- **name** (*str*)
- **plugin_cls** (type[P])

**list_default_plugins**()

Return a list of the names of available default plugins.

**Return type**
list[str]

**request_plugin**(*name*)

Return the plugin class which is registered under the name name, or raise NoPlugin if the name isn't available.

**Return type**
type[TypeVar(P)]

**Parameters**
**name** (*str*)

**copy**()

Return a copy of self.

**Return type**
*PluginPreset*[TypeVar(P)]

## Submodules

| | |
|---|---|
| *ansi* | |
| *autoimport* | |
| *bug_report* | |
| *hookset* | These classes perform some python magic that we use to implement the nesting of exploration technique methods. |
| *loggers* | |
| *picklable_lock* | |
| *plugins* | |
| *telemetry* | |
| *testing* | |
| *ux* | |

### 10.26.1 angr.misc.ansi

class angr.misc.ansi.**Color**

Bases: Enum

The basic ansi colors

**black = 30**

**red = 31**

**green = 32**

**yellow = 33**

**blue = 34**

**magenta = 35**

**cyan = 36**

**white = 37**

class angr.misc.ansi.**BackgroundColor**

Bases: Enum

**black = 40**

**red = 41**

**green = 42**

**yellow = 43**

**blue = 44**

**magenta = 45**

**cyan = 46**

**white = 47**

angr.misc.ansi.**color**(*c*, *bright*)

Return the ansi prefix using the given code Bright may not be used with a BackgroundColor

**Parameters**

- **c** (Color | BackgroundColor)
- **bright** (*bool*)

### 10.26.2 angr.misc.autoimport

angr.misc.autoimport.**auto_import_packages**(*base_module*, *base_path*, *ignore_dirs=()*, *ignore_files=()*, *scan_modules=True*)

angr.misc.autoimport.**auto_import_modules**(*base_module*, *base_path*, *ignore_files=()*, *filter_func=None*)

**Parameters**

**filter_func** (*Callable* | *None*)

angr.misc.autoimport.**filter_module**(*mod*, *type_req=None*, *subclass_req=None*)

angr.misc.autoimport.**auto_import_source_files**(*base_path*, *ignore_files=()*, *filter_func=None*)

**Parameters**
**filter_func** (*Callable | None*)

### 10.26.3 angr.misc.bug_report

angr.misc.bug_report.**get_venv**()

angr.misc.bug_report.**print_versions**()

angr.misc.bug_report.**print_git_info**(*dirname*)

angr.misc.bug_report.**print_system_info**()

angr.misc.bug_report.**print_native_info**()

angr.misc.bug_report.**bug_report**()

### 10.26.4 angr.misc.hookset

These classes perform some python magic that we use to implement the nesting of exploration technique methods. This process is formalized as a "hooking" of a python method - each exploration technique's methods "hooks" a method of the same name on the simulation manager class.

**class** angr.misc.hookset.**HookSet**

Bases: object

A HookSet is a static class that provides the capability to apply many hooks to an object.

**static install_hooks**(*target*, ***hooks*)

Given the target *target*, apply the hooks given as keyword arguments to it. If any targeted method has already been hooked, the hooks will not be overridden but will instead be pushed into a list of pending hooks. The final behavior should be that all hooks call each other in a nested stack.

**Parameters**

- **target** – Any object. Its methods named as keys in *hooks* will be replaced by *HookedMethod* objects.
- **hooks** – Any keywords will be interpreted as hooks to apply. Each method named will hooked with the corresponding function value.

**static remove_hooks**(*target*, ***hooks*)

Remove the given hooks from the given target.

**Parameters**

- **target** – The object from which to remove hooks. If all hooks are removed from a given method, the HookedMethod object will be removed and replaced with the original function.
- **hooks** – Any keywords will be interpreted as hooks to remove. You must provide the exact hook that was applied so that it can it can be identified for removal among any other hooks.

**static copy_hooks**(*source*, *target*, *domain*)

Copy the hooks from source onto target.

If the current callstack includes hooked methods from source, the already-called methods will not be included in the copy.

domain is a list of names that might be hooked.

**class** angr.misc.hookset.**HookedMethod**

Bases: object

HookedMethod is a callable object which provides a stack of nested hooks.

**Parameters**
**func** – The bottom-most function which provides the original functionality that is being hooked

**Variables**

- **func** – Same as the eponymous parameter
- **pending** – The stack of hooks that have yet to be called. When this object is called, it will pop the last function in this list and call it. The function should call this object again in order to request the functionality of the original method, at which point the pop-dispatch mechanism will run recursively until the stack is exhausted, at which point the original function will be called. When the call returns, the hook will be restored to the stack.

**__init__**(*func*)

**copy_to**(*new_func*)

### 10.26.5 angr.misc.loggers

**class** angr.misc.loggers.**Loggers**

Bases: object

Implements a loggers manager for angr.

**__init__**(*default_level=30*)

**default_level**

**profiling_enabled**

**handler**

**IN_SCOPE = ('angr', 'claripy', 'cle', 'pyvex', 'archinfo', 'tracer', 'driller', 'rex', 'patcherex', 'identifier')**

**load_all_loggers**()

A dumb and simple way to conveniently aggregate all loggers.

Adds attributes to this instance of each registered logger, replacing '.' with '_'

**enable_root_logger**()

Enable angr's default logger

**disable_root_logger**()

Disable angr's default logger

**static setall**(*level*)

**class** angr.misc.loggers.**CuteFormatter**

Bases: Formatter

A log formatter that can print log messages with colors.

angr.misc.loggers.**is_enabled_for**(*logger*, *level*)

### 10.26.6 angr.misc.picklable_lock

**class** angr.misc.picklable_lock.**PicklableLock**

Bases: object

Normal thread-locks are not pickleable. This provides a pickleable lock by mandating that the lock is unlocked during serialization.

**__init__**(**args*, ***kwargs*)

**acquire**(**args*, ***kwargs*)

**locked**()

**release**()

**class** angr.misc.picklable_lock.**PicklableRLock**

Bases: *PicklableLock*

Same as above, but uses RLock instead of Lock for locking. Note that RLock does not provide an interface to tell whether is it presently held by any thread, and thus this class will lie about whether it is locked.

**locked**()

### 10.26.7 angr.misc.plugins

**class** angr.misc.plugins.**PluginHub**

Bases: Generic

A plugin hub is an object which contains many plugins, as well as the notion of a "preset", or a backer that can provide default implementations of plugins which cater to a certain circumstance.

Objects in angr like the SimState, the Analyses hub, the SimEngine selector, etc all use this model to unify their mechanisms for automatically collecting and selecting components to use. If you're familiar with design patterns this is a configurable Strategy Pattern.

Each PluginHub subclass should have a corresponding Plugin subclass, and perhaps a PluginPreset subclass if it wants its presets to be able to specify anything more interesting than a list of defaults.

**__init__**()

**Return type**
None

**classmethod register_default**(*name*, *plugin_cls*, *preset='default'*)

**Return type**
None

**Parameters**

- **name** (*str*)
- **plugin_cls** (type)

- **preset** (*str*)

**classmethod register_preset**(*name*, *preset*)

Register a preset instance with the class of the hub it corresponds to. This allows individual plugin objects to automatically register themselves with a preset by using a classmethod of their own with only the name of the preset to register with.

**Return type**
None

**Parameters**

- **name** (*str*)
- **preset** (PluginPreset*[P]*)

**property plugin_preset: *PluginPreset*[P] | None**

Get the current active plugin preset

**property has_plugin_preset: bool**

Check whether or not there is a plugin preset in use on this hub right now

**use_plugin_preset**(*preset*)

Apply a preset to the hub. If there was a previously active preset, discard it.

Preset can be either the string name of a preset or a PluginPreset instance.

**Return type**
None

**Parameters**
**preset** (*str | PluginPreset[P]*)

**discard_plugin_preset**()

Discard the current active preset. Will release any active plugins that could have come from the old preset.

**Return type**
None

**get_plugin**(*name*)

Get the plugin named `name`. If no such plugin is currently active, try to activate a new one using the current preset.

**Return type**
TypeVar(P)

**Parameters**
**name** (*str*)

**has_plugin**(*name*)

Return whether or not a plugin with the name `name` is currently active.

**Return type**
bool

**Parameters**
**name** (*str*)

**register_plugin**(*name*, *plugin*)

Add a new plugin `plugin` with name `name` to the active plugins.

**Return type**
TypeVar(P)

**Parameters**

- **name** (*str*)
- **plugin** (*P*)

**release_plugin**(*name*)

Deactivate and remove the plugin with name `name`.

**Return type**
None

**Parameters**
**name** (*str*)

**class** angr.misc.plugins.**PluginPreset**

Bases: Generic

A plugin preset object contains a mapping from name to a plugin class. A preset can be active on a hub, which will cause it to handle requests for plugins which are not already present on the hub.

Unlike Plugins and PluginHubs, instances of PluginPresets are defined on the module level for individual presets. You should register the preset instance with a hub to allow plugins to easily add themselves to the preset without an explicit reference to the preset itself.

**__init__**()

**Return type**
None

**activate**(*hub*)

This method is called when the preset becomes active on a hub.

**Return type**
None

**Parameters**
**hub** (PluginHub*[P]*)

**deactivate**(*hub*)

This method is called when the preset is discarded from the hub.

**Return type**
None

**Parameters**
**hub** (PluginHub*[P]*)

**add_default_plugin**(*name*, *plugin_cls*)

Add a plugin to the preset.

**Return type**
None

**Parameters**

- **name** (*str*)
- **plugin_cls** (type*[P]*)

**list_default_plugins**()

Return a list of the names of available default plugins.

**Return type**
list[str]

**request_plugin**(*name*)

Return the plugin class which is registered under the name name, or raise NoPlugin if the name isn't available.

**Return type**
type[TypeVar(P)]

**Parameters**
**name** (*str*)

**copy**()

Return a copy of self.

**Return type**
*PluginPreset*[TypeVar(P)]

**class** angr.misc.plugins.**PluginVendor**

Bases: *PluginHub*, Generic

A specialized hub which serves only as a plugin vendor, never having any "active" plugins. It will directly return the plugins provided by the preset instead of instantiating them.

**release_plugin**(*name*)

This hub doesn't actually have any active plugins, so this is a no-op.

**Return type**
None

**Parameters**
**name** (*str*)

**register_plugin**(*name*, *plugin*)

This hub doesn't actually have any active plugins, so this is a no-op. We just return the plugin that was passed in to satisfy the return type of PluginHub.register_plugin.

**Return type**
TypeVar(P)

**Parameters**

- **name** (*str*)
- **plugin** (*P*)

**class** angr.misc.plugins.**VendorPreset**

Bases: *PluginPreset*, Generic

A specialized preset class for use with the PluginVendor.

### 10.26.8 angr.misc.telemetry

**class** angr.misc.telemetry.**Status**

Bases: object

Represents the status of a finished Span.

**Parameters**

- **status_code** (StatusCode) – The canonical status code that describes the result status of the operation.
- **description** (str | None) – An optional description of the status.

**__init__**(*status_code=StatusCode.UNSET*, *description=None*)

**Parameters**

- **status_code** (*StatusCode*)
- **description** (*str | None*)

**property status_code: StatusCode**

Represents the canonical status code of a finished Span.

**property description: str | None**

Status description

**property is_ok: bool**

Returns false if this represents an error, true otherwise.

**property is_unset: bool**

Returns true if unset, false otherwise.

**class** angr.misc.telemetry.**StatusCode**

Bases: Enum

Represents the canonical set of status codes of a finished Span.

**UNSET = 0**

The default status.

**OK = 1**

The operation has been validated by an Application developer or Operator to have completed successfully.

**ERROR = 2**

The operation contains an error.

angr.misc.telemetry.**get_current_span**(*context=None*)

Retrieve the current span.

**Parameters**

**context** (Context | None) – A Context object. If one is not passed, the default current context is used instead.

**Return type**

Span

**Returns**

The Span set in the context if it exists. INVALID_SPAN otherwise.

angr.misc.telemetry.**get_tracer**(*name*)

**Return type**

Tracer

**Parameters**

**name** (*str*)

### 10.26.9 angr.misc.testing

angr.misc.testing.**detect_test_env**()

### 10.26.10 angr.misc.ux

angr.misc.ux.**once**(*key*)

**Return type**
bool

**Parameters**
**key** (*str*)

angr.misc.ux.**deprecated**(*replacement=None*)

**Return type**
Callable[[Callable[[ParamSpec(P)], TypeVar(R)]], Callable[[ParamSpec(P)], TypeVar(R)]]

**Parameters**
**replacement** (*str | None*)

## 10.27 angr.procedures

#### Submodules

| | |
|---|---|
| *advapi32* | |
| *cgc* | These procedures implement system calls for the cgc DECREE platform |
| *definitions* | |
| *glibc* | These procedures implement internal functions for the GNU libc that commonly appears on linux systems |
| *gnulib* | These procedures implement functions described in the gnulib library: https://www.gnu.org/software/gnulib/ |
| *java* | |
| *java_io* | |
| *java_jni* | |
| *java_lang* | |
| *java_util* | |
| *libc* | These procedures implement functions described in the C standard library specification |
| *libstdcpp* | |
| *linux_kernel* | These procedures implement linux system calls or other misc functionality provided by the linux kernel |
| *linux_loader* | These function implement functions or other functionalities provided by the linux userspace dynamic loader |
| *msvcr* | These functions implement those found in the various versions of msvcr.dll (Microsoft Visual C/C++ Runtime) |
| *ntdll* | |
| *posix* | These procedures implement functions described in POSIX.1 |
| *procedure_dict* | |

continues on next page

Table 72 – continued from previous page

| | |
|---|---|
| *stubs* | These procedures are various misc stubs useful in analysis |
| *testing* | These procedures are used in the angr test suite. |
| *tracer* | These procedures implement system calls for the cgc DECREE platform, in a specific way for tracing. |
| *uclibc* | These procedures implement internal functions in the uClibc libc implementation |
| *win32* | These procedures implement functions from the win32 api, notably kernel32.dll |
| *win32_kernel* | These procedures implement functions from the Windows kernel. |
| *win_user32* | |

### 10.27.1 angr.procedures.advapi32

### 10.27.2 angr.procedures.cgc

These procedures implement system calls for the cgc DECREE platform

#### Submodules

| |
|---|
| *allocate* |
| *deallocate* |
| *fdwait* |
| *random* |
| *receive* |
| *transmit* |

#### angr.procedures.cgc.allocate

**class** angr.procedures.cgc.allocate.**allocate**

Bases: *SimProcedure*

#### angr.procedures.cgc.deallocate

**class** angr.procedures.cgc.deallocate.**deallocate**

Bases: *SimProcedure*

#### angr.procedures.cgc.fdwait

**class** angr.procedures.cgc.fdwait.**fdwait**

Bases: *SimProcedure*

#### angr.procedures.cgc.random

**class** angr.procedures.cgc.random.**random**

Bases: *SimProcedure*

#### angr.procedures.cgc.receive

**class** angr.procedures.cgc.receive.**receive**

Bases: *SimProcedure*

#### angr.procedures.cgc.transmit

**class** angr.procedures.cgc.transmit.**transmit**

Bases: *SimProcedure*

### 10.27.3 angr.procedures.definitions

**class** angr.procedures.definitions.**SimTypeCollection**

Bases: object

A type collection is the mechanism for describing types. Types in a type collection can be referenced using

**__init__**()

**set_names**(**names*)

**Parameters**
**names** (*str*)

**add**(*name*, *t*)

Add a type to the collection.

**Parameters**

- **name** (*str*) – Name of the type to add.
- **t** (*SimType*) – The SimType object to add to the collection.

**Return type**
None

**get**(*name*, *bottom_on_missing=False*, *memo=None*)

Get a SimType object from the collection as identified by the name.

**Parameters**

- **name** (*str*) – Name of the type to get.
- **bottom_on_missing** (*bool*) – Return a SimTypeBottom object if the required type does not exist.
- **memo** (*set[str] | None*) – A set of names that have been queried in this call chain (to prevent infinite recursion).

**Return type**
*SimType*

**Returns**
The SimType object.

**init_str**()

**Return type**
str

to_json(*types_as_string=False*)

**Return type**
dict[str, Any]

**Parameters**
**types_as_string** (*bool*)

classmethod from_json(*d*)

**Return type**
*SimTypeCollection*

**Parameters**
**d** (*dict[str, Any]*)

class angr.procedures.definitions.**SimLibrary**

Bases: object

A SimLibrary is the mechanism for describing a dynamic library's API, its functions and metadata.

Any instance of this class (or its subclasses) found in the `angr.procedures.definitions` package will be automatically picked up and added to `angr.SIM_LIBRARIES` via all its names.

**Variables**

- **fallback_cc** – A mapping from architecture to the default calling convention that should be used if no other information is present. Contains some sane defaults for linux.
- **fallback_proc** – A SimProcedure class that should be used to provide stub procedures. By default, `ReturnUnconstrained`.

__init__()

static from_json(*d*)

**Return type**
*SimLibrary*

**Parameters**
**d** (*dict[str, Any]*)

copy()

Make a copy of this SimLibrary, allowing it to be mutated without affecting the global version.

**Returns**
A new SimLibrary object with the same library references but different dict/list references

update(*other*)

Augment this SimLibrary with the information from another SimLibrary

**Parameters**
**other** (*SimLibrary*) – The other SimLibrary

property name

The first common name of this library, e.g. libc.so.6, or '??????' if none are known.

set_library_names(**names*)

Set some common names of this library by which it may be referred during linking

**Parameters**
**names** – Any number of string library names may be passed as varargs.

set_default_cc(*arch_name*, *cc_cls*)

Set the default calling convention used for this library under a given architecture

**Parameters**
**arch_name** – The string name of the architecture, i.e. the .name field from archinfo.

**Parm cc_cls**
The SimCC class (not an instance!) to use

set_non_returning(**names*)

Mark some functions in this class as never returning, i.e. loops forever or terminates execution

**Parameters**
**names** – Any number of string function names may be passed as varargs

set_prototype(*name*, *proto*)

Set the prototype of a function in the form of a SimTypeFunction containing argument and return types

**Parameters**

- **name** – The name of the function as a string
- **proto** (*SimTypeFunction*) – The prototype of the function as a SimTypeFunction

**Return type**
None

set_prototypes(*protos*)

Set the prototypes of many functions

**Parameters**
**protos** (dict[str, *SimTypeFunction*]) – Dictionary mapping function names to SimTypeFunction objects

**Return type**
None

set_c_prototype(*c_decl*)

Set the prototype of a function in the form of a C-style function declaration.

**Parameters**
**c_decl** (str) – The C-style declaration of the function.

**Return type**
tuple[str, *SimTypeFunction*]

**Returns**
A tuple of (function name, function prototype)

add(*name*, *proc_cls*, ***kwargs*)

Add a function implementation to the library.

**Parameters**

- **name** – The name of the function as a string
- **proc_cls** – The implementation of the function as a SimProcedure _class_, not instance
- **kwargs** – Any additional parameters to the procedure class constructor may be passed as kwargs

**add_all_from_dict**(*dictionary*, ***kwargs*)

Batch-add function implementations to the library.

**Parameters**

- **dictionary** – A mapping from name to procedure class, i.e. the first two arguments to add()
- **kwargs** – Any additional kwargs will be passed to the constructors of _each_ procedure class

**add_alias**(*name*, **alt_names*)

Add some duplicate names for a given function. The original function's implementation must already be registered.

**Parameters**

- **name** – The name of the function for which an implementation is already present
- **alt_names** – Any number of alternate names may be passed as varargs

**get**(*name*, *arch*)

Get an implementation of the given function specialized for the given arch, or a stub procedure if none exists.

**Parameters**

- **name** – The name of the function as a string
- **arch** – The architecure to use, as either a string or an archinfo.Arch instance

**Returns**

A SimProcedure instance representing the function as found in the library

**get_stub**(*name*, *arch*)

Get a stub procedure for the given function, regardless of if a real implementation is available. This will apply any metadata, such as a default calling convention or a function prototype.

By stub, we pretty much always mean a `ReturnUnconstrained` SimProcedure with the appropriate display name and metadata set. This will appear in `state.history.descriptions` as `<SimProcedure display_name (stub)>`

**Parameters**

- **name** – The name of the function as a string
- **arch** – The architecture to use, as either a string or an archinfo.Arch instance

**Returns**

A SimProcedure instance representing a plausable stub as could be found in the library.

**get_prototype**(*name*, *arch=None*, *deref=False*)

Get a prototype of the given function name, optionally specialize the prototype to a given architecture.

**Parameters**

- **name** (str) – Name of the function.
- **arch** – The architecture to specialize to.
- **deref** (bool) – True if any SimTypeRefs in the prototype should be dereferenced using library information.

**Return type**

*SimTypeFunction* | None

**Returns**
Prototype of the function, or None if the prototype does not exist.

`has_metadata`(*name*)

Check if a function has either an implementation or any metadata associated with it

**Parameters**
**name** – The name of the function as a string

**Returns**
A bool indicating if anything is known about the function

`has_implementation`(*name*)

Check if a function has an implementation associated with it

**Parameters**
**name** – The name of the function as a string

**Returns**
A bool indicating if an implementation of the function is available

`has_prototype`(*func_name*)

Check if a function has a prototype associated with it.

A prototype that is explicitly registered as `None` (i.e. the function is known but its signature is not) does not count: this keeps `has_prototype()` consistent with `get_prototype()`, which returns `None` for such entries.

**Parameters**
**func_name** (*str*) – The name of the function.

**Returns**
A bool indicating if a prototype of the function is available.

**Return type**
bool

`is_returning`(*name*)

Check if a function is known to return.

**Parameters**
**name** (`str`) – The name of the function.

**Return type**
`bool`

**Returns**
A bool indicating if the function is known to return or not.

`class angr.procedures.definitions.SimCppLibrary`

Bases: *SimLibrary*

SimCppLibrary is a specialized version of SimLibrary that will demangle C++ function names before looking for an implementation or prototype for it.

`get`(*name*, *arch*)

Get an implementation of the given function specialized for the given arch, or a stub procedure if none exists. Demangle the function name if it is a mangled C++ name.

**Parameters**

- **name** (*str*) – The name of the function as a string

- **arch** – The architecure to use, as either a string or an archinfo.Arch instance

**Returns**
A SimProcedure instance representing the function as found in the library

**get_stub**(*name*, *arch*)

Get a stub procedure for the given function, regardless of if a real implementation is available. This will apply any metadata, such as a default calling convention or a function prototype. Demangle the function name if it is a mangled C++ name.

**Parameters**

- **name** (*str*) – The name of the function as a string
- **arch** – The architecture to use, as either a string or an archinfo.Arch instance

**Returns**
A SimProcedure instance representing a plausable stub as could be found in the library.

**get_prototype**(*name*, *arch=None*, *deref=False*)

Get a prototype of the given function name, optionally specialize the prototype to a given architecture. The function name will be demangled first.

**Parameters**

- **name** (str) – Name of the function.
- **arch** – The architecture to specialize to.
- **deref** (bool) – True if any SimTypeRefs in the prototype should be dereferenced using library information.

**Return type**
*SimTypeFunction* | None

**Returns**
Prototype of the function, or None if the prototype does not exist.

**has_metadata**(*name*)

Check if a function has either an implementation or any metadata associated with it. Demangle the function name if it is a mangled C++ name.

**Parameters**
**name** – The name of the function as a string

**Returns**
A bool indicating if anything is known about the function

**has_implementation**(*name*)

Check if a function has an implementation associated with it. Demangle the function name if it is a mangled C++ name.

**Parameters**
**name** (*str*) – A mangled function name.

**Returns**
bool

**has_prototype**(*func_name*)

Check if a function has a prototype associated with it. Demangle the function name if it is a mangled C++ name.

**Parameters**
**name** (*str*) – A mangled function name.

**Returns**
bool

**class** angr.procedures.definitions.**SimSyscallLibrary**

Bases: *SimLibrary*

SimSyscallLibrary is a specialized version of SimLibrary for dealing not with a dynamic library's API but rather an operating system's syscall API. Because this interface is inherently lower-level than a dynamic library, many parts of this class has been changed to store data based on an "ABI name" (ABI = application binary interface, like an API but for when there's no programming language) instead of an architecture. An ABI name is just an arbitrary string with which a calling convention and a syscall numbering is associated.

All the SimLibrary methods for adding functions still work, but now there's an additional layer on top that associates them with numbers.

**__init__**()

**minimum_syscall_number**(*abi*)

**Parameters**
**abi** – The abi to evaluate

**Returns**
The smallest syscall number known for the given abi

**maximum_syscall_number**(*abi*)

**Parameters**
**abi** – The abi to evaluate

**Returns**
The largest syscall number known for the given abi

**add_number_mapping**(*abi*, *number*, *name*)

Associate a syscall number with the name of a function present in the underlying SimLibrary

**Parameters**

- **abi** – The abi for which this mapping applies
- **number** – The syscall number
- **name** – The name of the function

**add_number_mapping_from_dict**(*abi*, *mapping*)

Batch-associate syscall numbers with names of functions present in the underlying SimLibrary

**Parameters**

- **abi** – The abi for which this mapping applies
- **mapping** – A dict mapping syscall numbers to function names

**set_abi_cc**(*abi*, *cc_cls*)

Set the default calling convention for an abi

**Parameters**

- **abi** – The name of the abi
- **cc_cls** – A SimCC _class_, not an instance, that should be used for syscalls using the abi

**set_prototype**(*abi*, *name*, *proto*)

Set the prototype of a function in the form of a SimTypeFunction containing argument and return types

**Parameters**

- **abi** (str) – ABI of the syscall.
- **name** (str) – The name of the syscall as a string
- **proto** (*SimTypeFunction*) – The prototype of the syscall as a SimTypeFunction

**Return type**
None

**set_prototypes**(*abi*, *protos*)

Set the prototypes of many syscalls.

**Parameters**

- **abi** (str) – ABI of the syscalls.
- **protos** (dict[str, *SimTypeFunction*]) – Dictionary mapping syscall names to SimTypeFunction objects

**Return type**
None

**add_alias**(*name*, **alt_names*)

Add some duplicate names for a given function. The original function's implementation must already be registered.

**Parameters**

- **name** – The name of the function for which an implementation is already present
- **alt_names** – Any number of alternate names may be passed as varargs

**get**(*number*, *arch*, *abi_list=()*)

The get() function for SimSyscallLibrary looks a little different from its original version.

Instead of providing a name, you provide a number, and you additionally provide a list of abi names that are applicable. The first abi for which the number is present in the mapping will be chosen. This allows for the easy abstractions of architectures like ARM or MIPS linux for which there are many ABIs that can be used at any time by using syscall numbers from various ranges. If no abi knows about the number, the stub procedure with the name "sys_%d" will be used.

**Parameters**

- **number** – The syscall number
- **arch** – The architecture being worked with, as either a string name or an archinfo.Arch
- **abi_list** – A list of ABI names that could be used

**Returns**
A SimProcedure representing the implementation of the given syscall, or a stub if no implementation is available

**get_stub**(*number*, *arch*, *abi_list=()*)

Pretty much the intersection of SimLibrary.get_stub() and SimSyscallLibrary.get().

**Parameters**

- **number** – The syscall number

- **arch** – The architecture being worked with, as either a string name or an archinfo.Arch
- **abi_list** – A list of ABI names that could be used

**Returns**
A SimProcedure representing a plausable stub that could model the syscall

**get_prototype**(*abi*, *name*, *arch=None*, *deref=False*)

Get a prototype of the given syscall name and its ABI, optionally specialize the prototype to a given architecture.

**Parameters**

- **abi** (str) – ABI of the prototype to get.
- **name** (str) – Name of the syscall.
- **arch** – The architecture to specialize to.
- **deref** (bool) – True if any SimTypeRefs in the prototype should be dereferenced using library information.

**Return type**
*SimTypeFunction* | None

**Returns**
Prototype of the syscall, or None if the prototype does not exist.

**has_metadata**(*number*, *arch*, *abi_list=()*)

Pretty much the intersection of SimLibrary.has_metadata() and SimSyscallLibrary.get().

**Parameters**

- **number** – The syscall number
- **arch** – The architecture being worked with, as either a string name or an archinfo.Arch
- **abi_list** – A list of ABI names that could be used

**Returns**
A bool of whether or not any implementation or metadata is known about the given syscall

**has_implementation**(*number*, *arch*, *abi_list=()*)

Pretty much the intersection of SimLibrary.has_implementation() and SimSyscallLibrary.get().

**Parameters**

- **number** – The syscall number
- **arch** – The architecture being worked with, as either a string name or an archinfo.Arch
- **abi_list** – A list of ABI names that could be used

**Returns**
A bool of whether or not an implementation of the syscall is available

**has_prototype**(*abi*, *name*)

Check if a function has a prototype associated with it. Demangle the function name if it is a mangled C++ name.

Syscalls whose signature is unknown are registered with a `None` prototype (see, e.g., `capset` in `linux_kernel.py`). Those do not count as having a prototype: this keeps `has_prototype()` consistent with `get_prototype()`, which returns `None` for such entries.

**Parameters**

- **abi** (str) – Name of the ABI.
- **name** (str) – The syscall name.

**Return type**
bool

**Returns**
bool

angr.procedures.definitions.**load_type_collections**(*only=None*, *skip=None*)

**Return type**
None

angr.procedures.definitions.**load_win32_type_collections**()

**Return type**
None

angr.procedures.definitions.**load_external_definitions**()

Load library definitions from specific directories. By default it parses ANGR_EXTERNAL_DEFINITIONS_DIRS as a semicolon separated list of directory paths. Then it loads all .py files in each directory. These .py files should declare SimLibrary() objects and call .set_library_names() to register themselves in angr.SIM_LIBRARIES.

angr.procedures.definitions.**load_win32api_definitions**()

angr.procedures.definitions.**load_all_definitions**()

### Submodules


*cgc*
*gnulib*
*libstdcpp*
*linux_kernel*
*linux_loader*
*macho_libsystem*
*msvcr*
*parse_glibc*
*parse_syscalls_from_local_system*
*parse_win32json*
*types_stl*


#### angr.procedures.definitions.cgc

#### angr.procedures.definitions.gnulib

#### angr.procedures.definitions.libstdcpp

#### angr.procedures.definitions.linux_kernel

#### angr.procedures.definitions.linux_loader

#### angr.procedures.definitions.macho_libsystem

#### angr.procedures.definitions.msvcr

#### angr.procedures.definitions.parse_glibc

angr.procedures.definitions.parse_glibc.**main**()

### angr.procedures.definitions.parse_syscalls_from_local_system

angr.procedures.definitions.parse_syscalls_from_local_system.**parse_gdb_xml_file**(*fname*)

angr.procedures.definitions.parse_syscalls_from_local_system.**parse_unistd_include_header**(*header_path*)

angr.procedures.definitions.parse_syscalls_from_local_system.**dump_mapping**(*abi*, *mapping*)

angr.procedures.definitions.parse_syscalls_from_local_system.**main**()

### angr.procedures.definitions.parse_win32json

angr.procedures.definitions.parse_win32json.**is_anonymous_struct**(*s_name*)

**Return type**
bool

**Parameters**
**s_name** (*str*)

angr.procedures.definitions.parse_win32json.**get_angr_type_from_name**(*name*)

angr.procedures.definitions.parse_win32json.**get_typeref_if_available**(*t*)

**Return type**
*SimType*

**Parameters**
**t** (SimType)

angr.procedures.definitions.parse_win32json.**handle_json_type**(*t*, *anon_typelib*, *create_missing=False*)

**Parameters**

- **anon_typelib** (SimTypeCollection | *None*)
- **create_missing** (*bool*)

angr.procedures.definitions.parse_win32json.**create_angr_type_from_json**(*t*)

angr.procedures.definitions.parse_win32json.**do_it**(*in_dir*)

angr.procedures.definitions.parse_win32json.**main**()

### angr.procedures.definitions.types_stl

angr.procedures.definitions.types_stl.**EXACTDIV_ELEMENT_SIZES** = (3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511)

Element sizes for which an opaque `std::vector<T<N>>` class exists.

The std::vector<T>::size() exact-division patterns (see analyses/decompiler/known_patterns/std_vector_size.py) recover sizeof(T) from the compiler's magic constant but not the element type, so each supported size gets an opaque N-byte element class named `T<N>`. Only non-power-of-two sizes are listed: a power-of-two sizeof(T) divides with a bare shift and needs no magic constant (those are the named std::vector<short/int/long long> templates). The upper bound is empirical: real containers of large records exist (CryptoPP::ECPPoint is 112 bytes, and vectors of 100+ byte structs are common in C++ code), and covering only a handful of small sizes leaves the pattern silent on the code where the raw magic multiply is least readable.

angr.procedures.definitions.types_stl.**SHIFT_ELEMENT_SIZES** = (16, 32, 64, 128, 256, 512)

Power-of-two element sizes above 8, which divide with a bare shift but have no named C++ type of that width (short/int/long long cover 2/4/8).

Leaving these out mattered: sizeof(std::string) is 32 on libstdc++, and std::vector<std::string> is the single most common element type in the benchmark corpus. Without a T32 template its size() read is either unnamed or, worse, misidentified by whatever sub-pattern does match.

angr.procedures.definitions.types_stl.**OPAQUE_ELEMENT_SIZES** = (3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512)

Every size with an opaque `std::vector<T<N>>` class.

### 10.27.4 angr.procedures.glibc

These procedures implement internal functions for the GNU libc that commonly appears on linux systems

#### Submodules

| |
|---|
| *dynamic_loading* |
| *scanf* |
| *sscanf* |

#### angr.procedures.glibc.dynamic_loading

#### angr.procedures.glibc.scanf

#### angr.procedures.glibc.sscanf

### 10.27.5 angr.procedures.gnulib

These procedures implement functions described in the gnulib library: https://www.gnu.org/software/gnulib/

#### Submodules

*xalloc_die*
*xstrtol_fatal*

#### angr.procedures.gnulib.xalloc_die

**class** angr.procedures.gnulib.xalloc_die.**xalloc_die**

Bases: *SimProcedure*

xalloc_die

**NO_RET = True**

#### angr.procedures.gnulib.xstrtol_fatal

**class** angr.procedures.gnulib.xstrtol_fatal.**xstrtol_fatal**

Bases: *SimProcedure*

xstrtol_fatal

**NO_RET = True**

### 10.27.6 angr.procedures.java

**class** angr.procedures.java.**JavaSimProcedure**

Bases: *SimProcedure*

**property is_java**

#### Submodules

*unconstrained*

#### angr.procedures.java.unconstrained

**class** angr.procedures.java.unconstrained.**UnconstrainedMethod**

Bases: *JavaSimProcedure*

### 10.27.7 angr.procedures.java_io

#### Submodules

*read*
*write*

#### angr.procedures.java_io.read

**class** angr.procedures.java_io.read.**Read**

Bases: *JavaSimProcedure*

#### angr.procedures.java_io.write

class angr.procedures.java_io.write.**Write**

Bases: *JavaSimProcedure*

### 10.27.8 angr.procedures.java_jni

class angr.procedures.java_jni.**JNISimProcedure**

Bases: *SimProcedure*

Base SimProcedure class for JNI interface functions.

**return_ty:** str | None = None

**JNI_TRUE = 1**

**JNI_FALSE = 0**

#### Submodules


*array_operations*
*class_and_interface_operations*
*field_access*
*global_and_local_refs*
*method_calls*
*not_implemented*
*object_operations*
*string_operations*
*version_information*


#### angr.procedures.java_jni.array_operations

class angr.procedures.java_jni.array_operations.**GetArrayLength**

Bases: *JNISimProcedure*

**return_ty:** str | None = 'int'

class angr.procedures.java_jni.array_operations.**NewArray**

Bases: *JNISimProcedure*

**element_type:** str | None = None

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.array_operations.**NewBooleanArray**

Bases: *NewArray*

**element_type:** str | None = 'boolean'

class angr.procedures.java_jni.array_operations.**NewByteArray**

Bases: *NewArray*

**element_type:** str | None = 'byte'

class angr.procedures.java_jni.array_operations.**NewCharArray**

Bases: *NewArray*

element_type: str | None = 'char'

class angr.procedures.java_jni.array_operations.NewShortArray

Bases: NewArray

element_type: str | None = 'short'

class angr.procedures.java_jni.array_operations.NewIntArray

Bases: NewArray

element_type: str | None = 'int'

class angr.procedures.java_jni.array_operations.NewLongArray

Bases: NewArray

element_type: str | None = 'long'

class angr.procedures.java_jni.array_operations.NewObjectArray

Bases: JNISimProcedure

return_ty: str | None = 'reference'

class angr.procedures.java_jni.array_operations.GetObjectArrayElement

Bases: JNISimProcedure

return_ty: str | None = 'reference'

class angr.procedures.java_jni.array_operations.SetObjectArrayElement

Bases: JNISimProcedure

return_ty: str | None = 'reference'

class angr.procedures.java_jni.array_operations.GetArrayElements

Bases: JNISimProcedure

return_ty: str | None = 'reference'

class angr.procedures.java_jni.array_operations.ReleaseArrayElements

Bases: JNISimProcedure

return_ty: str | None = 'void'

JNI_COMMIT = 1

JNI_ABORT = 2

class angr.procedures.java_jni.array_operations.GetArrayRegion

Bases: JNISimProcedure

return_ty: str | None = 'void'

class angr.procedures.java_jni.array_operations.SetArrayRegion

Bases: JNISimProcedure

return_ty: str | None = 'void'

### angr.procedures.java_jni.class_and_interface_operations

class angr.procedures.java_jni.class_and_interface_operations.**GetSuperclass**

Bases: *JNISimProcedure*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.class_and_interface_operations.**FindClass**

Bases: *JNISimProcedure*

**return_ty:** str | None = 'reference'

### angr.procedures.java_jni.field_access

class angr.procedures.java_jni.field_access.**GetFieldID**

Bases: *JNISimProcedure*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.field_access.**GetStaticField**

Bases: *JNISimProcedure*

class angr.procedures.java_jni.field_access.**GetStaticBooleanField**

Bases: *GetStaticField*

**return_ty:** str | None = 'boolean'

class angr.procedures.java_jni.field_access.**GetStaticByteField**

Bases: *GetStaticField*

**return_ty:** str | None = 'byte'

class angr.procedures.java_jni.field_access.**GetStaticCharField**

Bases: *GetStaticField*

**return_ty:** str | None = 'char'

class angr.procedures.java_jni.field_access.**GetStaticShortField**

Bases: *GetStaticField*

**return_ty:** str | None = 'short'

class angr.procedures.java_jni.field_access.**GetStaticIntField**

Bases: *GetStaticField*

**return_ty:** str | None = 'int'

class angr.procedures.java_jni.field_access.**GetStaticLongField**

Bases: *GetStaticField*

**return_ty:** str | None = 'long'

class angr.procedures.java_jni.field_access.**GetStaticObjectField**

Bases: *GetStaticField*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.field_access.**SetStaticField**

Bases: *JNISimProcedure*

return_ty: str | None = 'void'

class angr.procedures.java_jni.field_access.GetField

Bases: *JNISimProcedure*

class angr.procedures.java_jni.field_access.GetBooleanField

Bases: *GetField*

return_ty: str | None = 'boolean'

class angr.procedures.java_jni.field_access.GetByteField

Bases: *GetField*

return_ty: str | None = 'byte'

class angr.procedures.java_jni.field_access.GetCharField

Bases: *GetField*

return_ty: str | None = 'char'

class angr.procedures.java_jni.field_access.GetShortField

Bases: *GetField*

return_ty: str | None = 'short'

class angr.procedures.java_jni.field_access.GetIntField

Bases: *GetField*

return_ty: str | None = 'int'

class angr.procedures.java_jni.field_access.GetLongField

Bases: *GetField*

return_ty: str | None = 'long'

class angr.procedures.java_jni.field_access.GetObjectField

Bases: *GetField*

return_ty: str | None = 'reference'

class angr.procedures.java_jni.field_access.SetField

Bases: *JNISimProcedure*

return_ty: str | None = 'void'

### angr.procedures.java_jni.global_and_local_refs

class angr.procedures.java_jni.global_and_local_refs.NewGlobalRef

Bases: *JNISimProcedure*

return_ty: str | None = 'reference'

class angr.procedures.java_jni.global_and_local_refs.DeleteGlobalRef

Bases: *JNISimProcedure*

return_ty: str | None = 'void'

class angr.procedures.java_jni.global_and_local_refs.NewLocalRef

Bases: *JNISimProcedure*

return_ty: str | None = 'reference'

class angr.procedures.java_jni.global_and_local_refs.DeleteLocalRef

Bases: *JNISimProcedure*

return_ty: str | None = 'void'

### angr.procedures.java_jni.method_calls

class angr.procedures.java_jni.method_calls.GetMethodID

Bases: *JNISimProcedure*

return_ty: str | None = 'reference'

class angr.procedures.java_jni.method_calls.CallMethodBase

Bases: *JNISimProcedure*

return_ty: str | None = None

class angr.procedures.java_jni.method_calls.CallMethod

Bases: *CallMethodBase*

return_from_invocation(*ptr_env*, *obj_*, *method_id_*)

class angr.procedures.java_jni.method_calls.CallMethodA

Bases: *CallMethodBase*

return_from_invocation(*ptr_env*, *obj_*, *method_id_*, *ptr_args*)

class angr.procedures.java_jni.method_calls.CallObjectMethod

Bases: *CallMethod*

return_ty: str | None = 'reference'

class angr.procedures.java_jni.method_calls.CallBooleanMethod

Bases: *CallMethod*

return_ty: str | None = 'boolean'

class angr.procedures.java_jni.method_calls.CallByteMethod

Bases: *CallMethod*

return_ty: str | None = 'byte'

class angr.procedures.java_jni.method_calls.CallCharMethod

Bases: *CallMethod*

return_ty: str | None = 'char'

class angr.procedures.java_jni.method_calls.CallShortMethod

Bases: *CallMethod*

return_ty: str | None = 'short'

class angr.procedures.java_jni.method_calls.CallIntMethod

Bases: *CallMethod*

return_ty: str | None = 'int'

class angr.procedures.java_jni.method_calls.**CallLongMethod**

Bases: *CallMethod*

**return_ty:** str | None = 'long'

class angr.procedures.java_jni.method_calls.**CallVoidMethod**

Bases: *CallMethod*

**return_ty:** str | None = 'void'

class angr.procedures.java_jni.method_calls.**CallObjectMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.method_calls.**CallBooleanMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'boolean'

class angr.procedures.java_jni.method_calls.**CallByteMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'byte'

class angr.procedures.java_jni.method_calls.**CallCharMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'char'

class angr.procedures.java_jni.method_calls.**CallShortMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'short'

class angr.procedures.java_jni.method_calls.**CallIntMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'int'

class angr.procedures.java_jni.method_calls.**CallLongMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'long'

class angr.procedures.java_jni.method_calls.**CallVoidMethodA**

Bases: *CallMethodA*

**return_ty:** str | None = 'void'

class angr.procedures.java_jni.method_calls.**CallNonvirtualMethod**

Bases: *CallMethodBase*

**return_from_invocation**(*ptr_env*, *obj_*, *class_*, *method_id_*)

class angr.procedures.java_jni.method_calls.**CallNonvirtualMethodA**

Bases: *CallMethodBase*

**return_from_invocation**(*ptr_env*, *obj_*, *method_id_*, *ptr_args*)

class angr.procedures.java_jni.method_calls.**CallNonvirtualObjectMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.method_calls.**CallNonvirtualBooleanMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'boolean'

class angr.procedures.java_jni.method_calls.**CallNonvirtualByteMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'byte'

class angr.procedures.java_jni.method_calls.**CallNonvirtualCharMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'char'

class angr.procedures.java_jni.method_calls.**CallNonvirtualShortMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'short'

class angr.procedures.java_jni.method_calls.**CallNonvirtualIntMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'int'

class angr.procedures.java_jni.method_calls.**CallNonvirtualLongMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'long'

class angr.procedures.java_jni.method_calls.**CallNonvirtualVoidMethod**

Bases: *CallNonvirtualMethod*

**return_ty:** str | None = 'void'

class angr.procedures.java_jni.method_calls.**CallNonvirtualObjectMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.method_calls.**CallNonvirtualBooleanMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'boolean'

class angr.procedures.java_jni.method_calls.**CallNonvirtualByteMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'byte'

class angr.procedures.java_jni.method_calls.**CallNonvirtualCharMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'char'

class angr.procedures.java_jni.method_calls.**CallNonvirtualShortMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'short'

class angr.procedures.java_jni.method_calls.**CallNonvirtualIntMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'int'

class angr.procedures.java_jni.method_calls.**CallNonvirtualLongMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'long'

class angr.procedures.java_jni.method_calls.**CallNonvirtualVoidMethodA**

Bases: *CallNonvirtualMethodA*

**return_ty:** str | None = 'void'

class angr.procedures.java_jni.method_calls.**CallStaticMethod**

Bases: *CallMethodBase*

**return_from_invocation**(*ptr_env*, *class_*, *method_id_*)

class angr.procedures.java_jni.method_calls.**CallStaticMethodA**

Bases: *CallMethodBase*

**return_from_invocation**(*ptr_env*, *class_*, *method_id_*, *ptr_args*)

class angr.procedures.java_jni.method_calls.**CallStaticObjectMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.method_calls.**CallStaticBooleanMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'boolean'

class angr.procedures.java_jni.method_calls.**CallStaticByteMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'byte'

class angr.procedures.java_jni.method_calls.**CallStaticCharMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'char'

class angr.procedures.java_jni.method_calls.**CallStaticShortMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'short'

class angr.procedures.java_jni.method_calls.**CallStaticIntMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'int'

class angr.procedures.java_jni.method_calls.**CallStaticLongMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'long'

class angr.procedures.java_jni.method_calls.**CallStaticVoidMethod**

Bases: *CallStaticMethod*

**return_ty:** str | None = 'void'

class angr.procedures.java_jni.method_calls.**CallStaticObjectMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.method_calls.**CallStaticBooleanMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'boolean'

class angr.procedures.java_jni.method_calls.**CallStaticByteMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'byte'

class angr.procedures.java_jni.method_calls.**CallStaticCharMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'char'

class angr.procedures.java_jni.method_calls.**CallStaticShortMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'short'

class angr.procedures.java_jni.method_calls.**CallStaticIntMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'int'

class angr.procedures.java_jni.method_calls.**CallStaticLongMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'long'

class angr.procedures.java_jni.method_calls.**CallStaticVoidMethodA**

Bases: *CallStaticMethodA*

**return_ty:** str | None = 'void'

### angr.procedures.java_jni.not_implemented

class angr.procedures.java_jni.not_implemented.**UnsupportedJNIFunction**

Bases: *JNISimProcedure*

**return_ty:** str | None = 'void'

### angr.procedures.java_jni.object_operations

class angr.procedures.java_jni.object_operations.**GetObjectClass**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.object_operations.**AllocObject**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.object_operations.**NewObject**
Bases: *CallMethodBase*

**return_ty:** str | None = 'reference'

**obj** = None

**local_vars:** tuple[str, ...] = ('obj',)

**return_from_invocation**(*ptr_env*, *obj_class_*, *method_id_*)

class angr.procedures.java_jni.object_operations.**IsInstanceOf**
Bases: *CallMethodBase*

**return_ty:** str | None = 'boolean'

class angr.procedures.java_jni.object_operations.**IsSameObject**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'boolean'

### angr.procedures.java_jni.string_operations

class angr.procedures.java_jni.string_operations.**GetStringUTFChars**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.string_operations.**ReleaseStringUTFChars**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'void'

class angr.procedures.java_jni.string_operations.**NewStringUTF**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'reference'

class angr.procedures.java_jni.string_operations.**GetStringUTFLength**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'int'

### angr.procedures.java_jni.version_information

class angr.procedures.java_jni.version_information.**GetVersion**
Bases: *JNISimProcedure*

**return_ty:** str | None = 'int'

### 10.27.9 angr.procedures.java_lang

#### Submodules


| |
|---|
| *character* |
| *double* |
| *exit* |
| *getsimplename* |
| *integer* |
| *load_library* |
| *math* |
| *string* |
| *stringbuilder* |
| *system* |


#### angr.procedures.java_lang.character

**class** angr.procedures.java_lang.character.**CharacterIsDigit**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.character.**CharacterIsSpaceChar**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.double

**class** angr.procedures.java_lang.double.**ParseDouble**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.exit

**class** angr.procedures.java_lang.exit.**Exit**

Bases: *JavaSimProcedure*

**NO_RET = True**

#### angr.procedures.java_lang.getsimplename

**class** angr.procedures.java_lang.getsimplename.**GetSimpleName**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.integer

**class** angr.procedures.java_lang.integer.**IntToInteger**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.integer.**IntegerToInt**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.integer.**IntegerToString**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.integer.**IntegerValueOf**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.load_library

**class** angr.procedures.java_lang.load_library.**LoadLibrary**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.math

**class** angr.procedures.java_lang.math.**Random**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.string

**class** angr.procedures.java_lang.string.**StringConcat**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.string.**StringEquals**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.string.**StringSplit**

Bases: *JavaSimProcedure*

**generate_symbolic_array**(*state*, *max_length=1000*)

**class** angr.procedures.java_lang.string.**StringLength**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.string.**StringCharAt**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.stringbuilder

**class** angr.procedures.java_lang.stringbuilder.**StringBuilderInit**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_lang.stringbuilder.**StringBuilderAppend**

Bases: *JavaSimProcedure*

#### angr.procedures.java_lang.system

**class** angr.procedures.java_lang.system.**SystemCurrentTimeMillis**

Bases: *JavaSimProcedure*

### 10.27.10 angr.procedures.java_util

#### Submodules

| |
|---|
| *collection* |
| *iterator* |
| *list* |
| *map* |
| *random* |
| *scanner_nextline* |

### angr.procedures.java_util.collection

**class** angr.procedures.java_util.collection.**GetIterator**

Bases: *JavaSimProcedure*

### angr.procedures.java_util.iterator

**class** angr.procedures.java_util.iterator.**IteratorHasNext**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.iterator.**IteratorNext**

Bases: *JavaSimProcedure*

### angr.procedures.java_util.list

**class** angr.procedures.java_util.list.**ListInit**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.list.**ListAdd**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.list.**ListGet**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.list.**ListGetFirst**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.list.**ListSize**

Bases: *JavaSimProcedure*

### angr.procedures.java_util.map

angr.procedures.java_util.map.**get_map_key**(*state*, *key_ref*)

**class** angr.procedures.java_util.map.**MapInit**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.map.**MapPut**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.map.**MapGet**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.map.**MapSize**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.map.**MapContainsKey**

Bases: *JavaSimProcedure*

**class** angr.procedures.java_util.map.**MapKeySet**

Bases: *JavaSimProcedure*

### angr.procedures.java_util.random

**class** angr.procedures.java_util.random.**NextInt**

Bases: *JavaSimProcedure*

#### angr.procedures.java_util.scanner_nextline

**class** angr.procedures.java_util.scanner_nextline.**ScannerNextLine**

Bases: *JavaSimProcedure*

### 10.27.11 angr.procedures.libc

These procedures implement functions described in the C standard library specification

#### Submodules

| |
|---|
| *abort* |
| *access* |
| *atoi* |
| *atol* |
| *calloc* |
| *closelog* |
| *err* |
| *error* |
| *exit* |
| *fclose* |
| *feof* |
| *fflush* |
| *fgetc* |
| *fgets* |
| *fopen* |
| *fprintf* |
| *fputc* |
| *fputs* |
| *fread* |
| *free* |
| *fscanf* |
| *fseek* |
| *ftell* |
| *fwrite* |
| *getchar* |
| *getdelim* |
| *getegid* |
| *geteuid* |
| *getgid* |
| *gets* |
| *getuid* |
| *isalnum* |
| *isalpha* |
| *isascii* |
| *isblank* |
| *iscntrl* |
| *isdigit* |
| *isgraph* |
| *islower* |
| *isprint* |
| *ispunct* |
| *isspace* |

continues on next page

Table 82 – continued from previous page

| |
|---|
| *isupper* |
| *isxdigit* |
| *malloc* |
| *memccpy* |
| *memchr* |
| *memcmp* |
| *memcpy* |
| *memmove* |
| *mempcpy* |
| *memset* |
| *openlog* |
| *perror* |
| *printf* |
| *putchar* |
| *puts* |
| *rand* |
| *realloc* |
| *rewind* |
| *scanf* |
| *setbuf* |
| *setvbuf* |
| *snprintf* |
| *sprintf* |
| *srand* |
| *sscanf* |
| *stpcpy* |
| *strcat* |
| *strchr* |
| *strcmp* |
| *strcoll* |
| *strcpy* |
| *strlen* |
| *strncat* |
| *strncmp* |
| *strncpy* |
| *strnlen* |
| *strstr* |
| *strtok* |
| *strtol* |
| *strtoul* |
| *strxfrm* |
| *system* |
| *time* |
| *tmpnam* |
| *tolower* |
| *toupper* |
| *ungetc* |
| *vsnprintf* |
| *wchar* |

### angr.procedures.libc.abort

class angr.procedures.libc.abort.**abort**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.libc.access

class angr.procedures.libc.access.**access**

Bases: *SimProcedure*

### angr.procedures.libc.atoi

class angr.procedures.libc.atoi.**atoi**

Bases: *SimProcedure*

### angr.procedures.libc.atol

class angr.procedures.libc.atol.**atol**

Bases: *SimProcedure*

### angr.procedures.libc.calloc

class angr.procedures.libc.calloc.**calloc**

Bases: *SimProcedure*

### angr.procedures.libc.closelog

class angr.procedures.libc.closelog.**closelog**

Bases: *SimProcedure*

### angr.procedures.libc.err

class angr.procedures.libc.err.**err**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.libc.error

class angr.procedures.libc.error.**error**

Bases: *SimProcedure*

**NO_RET = None**

**DYNAMIC_RET = True**

### angr.procedures.libc.exit

class angr.procedures.libc.exit.**exit**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.libc.fclose

class angr.procedures.libc.fclose.**fclose**

Bases: *SimProcedure*

### angr.procedures.libc.feof

class angr.procedures.libc.feof.**feof**

Bases: *SimProcedure*

angr.procedures.libc.feof.**feof_unlocked**

alias of *feof*

### angr.procedures.libc.fflush

class angr.procedures.libc.fflush.**fflush**

Bases: *SimProcedure*

angr.procedures.libc.fflush.**fflush_unlocked**

alias of *fflush*

### angr.procedures.libc.fgetc

class angr.procedures.libc.fgetc.**fgetc**

Bases: *SimProcedure*

angr.procedures.libc.fgetc.**getc**

alias of *fgetc*

angr.procedures.libc.fgetc.**fgetc_unlocked**

alias of *fgetc*

angr.procedures.libc.fgetc.**getc_unlocked**

alias of *fgetc*

### angr.procedures.libc.fgets

class angr.procedures.libc.fgets.**fgets**

Bases: *SimProcedure*

angr.procedures.libc.fgets.**fgets_unlocked**

alias of *fgets*

### angr.procedures.libc.fopen

angr.procedures.libc.fopen.**mode_to_flag**(*mode*)

class angr.procedures.libc.fopen.**fopen**

Bases: *SimProcedure*

### angr.procedures.libc.fprintf

class angr.procedures.libc.fprintf.**fprintf**

Bases: *FormatParser*

### angr.procedures.libc.fputc

**class** angr.procedures.libc.fputc.**fputc**

Bases: *SimProcedure*

angr.procedures.libc.fputc.**putc**

alias of *fputc*

angr.procedures.libc.fputc.**fputc_unlocked**

alias of *fputc*

angr.procedures.libc.fputc.**putc_unlocked**

alias of *fputc*

### angr.procedures.libc.fputs

**class** angr.procedures.libc.fputs.**fputs**

Bases: *SimProcedure*

angr.procedures.libc.fputs.**fputs_unlocked**

alias of *fputs*

### angr.procedures.libc.fread

**class** angr.procedures.libc.fread.**fread**

Bases: *SimProcedure*

angr.procedures.libc.fread.**fread_unlocked**

alias of *fread*

### angr.procedures.libc.free

**class** angr.procedures.libc.free.**free**

Bases: *SimProcedure*

### angr.procedures.libc.fscanf

**class** angr.procedures.libc.fscanf.**fscanf**

Bases: *ScanfFormatParser*

### angr.procedures.libc.fseek

**class** angr.procedures.libc.fseek.**fseek**

Bases: *SimProcedure*

angr.procedures.libc.fseek.**fseeko**

alias of *fseek*

### angr.procedures.libc.ftell

**class** angr.procedures.libc.ftell.**ftell**

Bases: *SimProcedure*

angr.procedures.libc.ftell.**ftello**

alias of *ftell*

### angr.procedures.libc.fwrite

class angr.procedures.libc.fwrite.**fwrite**

Bases: *SimProcedure*

angr.procedures.libc.fwrite.**fwrite_unlocked**

alias of *fwrite*

### angr.procedures.libc.getchar

class angr.procedures.libc.getchar.**getchar**

Bases: *SimProcedure*

angr.procedures.libc.getchar.**getchar_unlocked**

alias of *getchar*

### angr.procedures.libc.getdelim

### angr.procedures.libc.getegid

class angr.procedures.libc.getegid.**getegid**

Bases: *SimProcedure*

### angr.procedures.libc.geteuid

class angr.procedures.libc.geteuid.**geteuid**

Bases: *SimProcedure*

### angr.procedures.libc.getgid

class angr.procedures.libc.getgid.**getgid**

Bases: *SimProcedure*

### angr.procedures.libc.gets

class angr.procedures.libc.gets.**gets**

Bases: *SimProcedure*

### angr.procedures.libc.getuid

class angr.procedures.libc.getuid.**getuid**

Bases: *SimProcedure*

### angr.procedures.libc.isalnum

class angr.procedures.libc.isalnum.**isalnum**

Bases: *SimProcedure*

### angr.procedures.libc.isalpha

class angr.procedures.libc.isalpha.**isalpha**

Bases: *SimProcedure*

### angr.procedures.libc.isascii

class angr.procedures.libc.isascii.**isascii**

Bases: *SimProcedure*

### angr.procedures.libc.isblank

class angr.procedures.libc.isblank.**isblank**

Bases: *SimProcedure*

### angr.procedures.libc.iscntrl

class angr.procedures.libc.iscntrl.**iscntrl**

Bases: *SimProcedure*

### angr.procedures.libc.isdigit

class angr.procedures.libc.isdigit.**isdigit**

Bases: *SimProcedure*

### angr.procedures.libc.isgraph

class angr.procedures.libc.isgraph.**isgraph**

Bases: *SimProcedure*

### angr.procedures.libc.islower

class angr.procedures.libc.islower.**islower**

Bases: *SimProcedure*

### angr.procedures.libc.isprint

class angr.procedures.libc.isprint.**isprint**

Bases: *SimProcedure*

### angr.procedures.libc.ispunct

class angr.procedures.libc.ispunct.**ispunct**

Bases: *SimProcedure*

### angr.procedures.libc.isspace

class angr.procedures.libc.isspace.**isspace**

Bases: *SimProcedure*

### angr.procedures.libc.isupper

class angr.procedures.libc.isupper.**isupper**

Bases: *SimProcedure*

### angr.procedures.libc.isxdigit

class angr.procedures.libc.isxdigit.**isxdigit**

Bases: *SimProcedure*

### angr.procedures.libc.malloc

class angr.procedures.libc.malloc.**malloc**

Bases: *SimProcedure*

### angr.procedures.libc.memccpy

class angr.procedures.libc.memccpy.**memccpy**

Bases: *SimProcedure*

### angr.procedures.libc.memchr

class angr.procedures.libc.memchr.**memchr**

Bases: *SimProcedure*

### angr.procedures.libc.memcmp

class angr.procedures.libc.memcmp.**memcmp**

Bases: *SimProcedure*

### angr.procedures.libc.memcpy

class angr.procedures.libc.memcpy.**memcpy**

Bases: *SimProcedure*

### angr.procedures.libc.memmove

class angr.procedures.libc.memmove.**memmove**

Bases: *SimProcedure*

### angr.procedures.libc.mempcpy

class angr.procedures.libc.mempcpy.**mempcpy**

Bases: *SimProcedure*

### angr.procedures.libc.memset

class angr.procedures.libc.memset.**memset**

Bases: *SimProcedure*

### angr.procedures.libc.openlog

class angr.procedures.libc.openlog.**openlog**

Bases: *SimProcedure*

### angr.procedures.libc.perror

class angr.procedures.libc.perror.**perror**

Bases: *SimProcedure*

### angr.procedures.libc.printf

class angr.procedures.libc.printf.**printf**

Bases: *FormatParser*

#### angr.procedures.libc.putchar

class angr.procedures.libc.putchar.**putchar**

Bases: *SimProcedure*

#### angr.procedures.libc.puts

class angr.procedures.libc.puts.**puts**

Bases: *SimProcedure*

#### angr.procedures.libc.rand

class angr.procedures.libc.rand.**rand**

Bases: *SimProcedure*

#### angr.procedures.libc.realloc

class angr.procedures.libc.realloc.**realloc**

Bases: *SimProcedure*

#### angr.procedures.libc.rewind

class angr.procedures.libc.rewind.**rewind**

Bases: *SimProcedure*

#### angr.procedures.libc.scanf

class angr.procedures.libc.scanf.**scanf**

Bases: *ScanfFormatParser*

#### angr.procedures.libc.setbuf

class angr.procedures.libc.setbuf.**setbuf**

Bases: *SimProcedure*

#### angr.procedures.libc.setvbuf

class angr.procedures.libc.setvbuf.**setvbuf**

Bases: *SimProcedure*

#### angr.procedures.libc.snprintf

class angr.procedures.libc.snprintf.**snprintf**

Bases: *FormatParser*

#### angr.procedures.libc.sprintf

class angr.procedures.libc.sprintf.**sprintf**

Bases: *FormatParser*

#### angr.procedures.libc.srand

class angr.procedures.libc.srand.**srand**

Bases: *SimProcedure*

### angr.procedures.libc.sscanf

class angr.procedures.libc.sscanf.**sscanf**

Bases: *ScanfFormatParser*

### angr.procedures.libc.stpcpy

class angr.procedures.libc.stpcpy.**stpcpy**

Bases: *SimProcedure*

stpcpy

### angr.procedures.libc.strcat

class angr.procedures.libc.strcat.**strcat**

Bases: *SimProcedure*

### angr.procedures.libc.strchr

class angr.procedures.libc.strchr.**strchr**

Bases: *SimProcedure*

### angr.procedures.libc.strcmp

class angr.procedures.libc.strcmp.**strcmp**

Bases: *SimProcedure*

### angr.procedures.libc.strcoll

class angr.procedures.libc.strcoll.**strcoll**

Bases: *SimProcedure*

### angr.procedures.libc.strcpy

class angr.procedures.libc.strcpy.**strcpy**

Bases: *SimProcedure*

### angr.procedures.libc.strlen

class angr.procedures.libc.strlen.**strlen**

Bases: *SimProcedure*

**max_null_index = None**

### angr.procedures.libc.strncat

class angr.procedures.libc.strncat.**strncat**

Bases: *SimProcedure*

### angr.procedures.libc.strncmp

class angr.procedures.libc.strncmp.**strncmp**

Bases: *SimProcedure*

### angr.procedures.libc.strncpy

**class** angr.procedures.libc.strncpy.**strncpy**

Bases: *SimProcedure*

### angr.procedures.libc.strnlen

**class** angr.procedures.libc.strnlen.**strnlen**

Bases: *SimProcedure*

### angr.procedures.libc.strstr

**class** angr.procedures.libc.strstr.**strstr**

Bases: *SimProcedure*

### angr.procedures.libc.strtok

**class** angr.procedures.libc.strtok.**strtok**

Bases: *SimProcedure*

**KEY = 'strtok_save_ptr'**

### angr.procedures.libc.strtol

**class** angr.procedures.libc.strtol.**strtol**

Bases: *SimProcedure*

**static strtol_inner**(*s*, *state*, *region*, *base*, *signed*, *read_length=None*)

Inner parser for strtol-style functions.

Note: all numbers may start with +/- and base 16 may start with 0x.

**Parameters**

- **s** – the string address/offset
- **state** – SimState
- **region** – memory, file, etc
- **base** – the base to use to interpret the number
- **signed** – boolean, true means the result will be signed, otherwise unsigned
- **read_length** – int, the number of bytes parsed in strtol

**Returns**

`(expression, value, num_bytes)` where `expression` is a symbolic boolean indicating success, `value` is the returned value (set to 0 on failure, or min/max on overflow), and `num_bytes` is the number of bytes read in the string.

### angr.procedures.libc.strtoul

**class** angr.procedures.libc.strtoul.**strtoul**

Bases: *SimProcedure*

#### angr.procedures.libc.strxfrm

class angr.procedures.libc.strxfrm.**strxfrm**

Bases: *SimProcedure*

Transform string for locale-aware comparison.

#### angr.procedures.libc.system

class angr.procedures.libc.system.**system**

Bases: *SimProcedure*

#### angr.procedures.libc.time

class angr.procedures.libc.time.**time**

Bases: *SimProcedure*

#### angr.procedures.libc.tmpnam

class angr.procedures.libc.tmpnam.**tmpnam**

Bases: *SimProcedure*

#### angr.procedures.libc.tolower

class angr.procedures.libc.tolower.**tolower**

Bases: *SimProcedure*

#### angr.procedures.libc.toupper

class angr.procedures.libc.toupper.**toupper**

Bases: *SimProcedure*

#### angr.procedures.libc.ungetc

class angr.procedures.libc.ungetc.**ungetc**

Bases: *SimProcedure*

#### angr.procedures.libc.vsnprintf

class angr.procedures.libc.vsnprintf.**vsnprintf**

Bases: *SimProcedure*

#### angr.procedures.libc.wchar

class angr.procedures.libc.wchar.**wcscmp**

Bases: *SimProcedure*

class angr.procedures.libc.wchar.**wcscasecmp**

Bases: *SimProcedure*

### 10.27.12 angr.procedures.libstdcpp

#### Submodules

*std____throw_bad_alloc*
*std____throw_bad_cast*
*std____throw_length_error*
*std____throw_logic_error*
*std__terminate*

#### angr.procedures.libstdcpp.std____throw_bad_alloc

**class** angr.procedures.libstdcpp.std____throw_bad_alloc.**std____throw_bad_cast**

Bases: *SimProcedure*

**NO_RET = True**

**ALT_NAMES = ('std::__throw_bad_alloc()',)**

#### angr.procedures.libstdcpp.std____throw_bad_cast

**class** angr.procedures.libstdcpp.std____throw_bad_cast.**std____throw_bad_cast**

Bases: *SimProcedure*

**NO_RET = True**

**ALT_NAMES = ('std::__throw_bad_cast()',)**

#### angr.procedures.libstdcpp.std____throw_length_error

**class** angr.procedures.libstdcpp.std____throw_length_error.**std____throw_logic_error**

Bases: *SimProcedure*

**NO_RET = True**

**ALT_NAMES = ('std::__throw_length_error(char const*)',)**

#### angr.procedures.libstdcpp.std____throw_logic_error

**class** angr.procedures.libstdcpp.std____throw_logic_error.**std____throw_logic_error**

Bases: *SimProcedure*

**NO_RET = True**

**ALT_NAMES = ('std::__throw_logic_error(char const*)',)**

#### angr.procedures.libstdcpp.std__terminate

**class** angr.procedures.libstdcpp.std__terminate.**std__terminate**

Bases: *SimProcedure*

**NO_RET = True**

**ALT_NAMES = ('std::terminate()',)**

### 10.27.13 angr.procedures.linux_kernel

These procedures implement linux system calls or other misc functionality provided by the linux kernel

### Submodules


access
arch_prctl
arm_user_helpers
brk
cwd
fstat
fstat64
futex
getegid
geteuid
getgid
getpid
getrlimit
gettid
getuid
iovec
lseek
mmap
mprotect
munmap
openat
set_tid_address
sigaction
sigprocmask
stat
sysinfo
tgkill
time
uid
uname
unlink
vsyscall


#### angr.procedures.linux_kernel.access

**class** angr.procedures.linux_kernel.access.**access**

Bases: *SimProcedure*

#### angr.procedures.linux_kernel.arch_prctl

**class** angr.procedures.linux_kernel.arch_prctl.**arch_prctl**

Bases: *SimProcedure*

Sets the architecture specific thread state based on the subfunction selected using the 'code' parameter. This syscall is only present on x86_64 linux

#### angr.procedures.linux_kernel.arm_user_helpers

#### angr.procedures.linux_kernel.brk

**class** angr.procedures.linux_kernel.brk.**brk**

Bases: *SimProcedure*

This implements the brk system call.

### angr.procedures.linux_kernel.cwd

**class** angr.procedures.linux_kernel.cwd.**getcwd**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.cwd.**chdir**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.fstat

**class** angr.procedures.linux_kernel.fstat.**fstat**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.fstat64

**class** angr.procedures.linux_kernel.fstat64.**fstat64**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.futex

**class** angr.procedures.linux_kernel.futex.**futex**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.getegid

**class** angr.procedures.linux_kernel.getegid.**getegid**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.getegid.**getegid32**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.geteuid

**class** angr.procedures.linux_kernel.geteuid.**geteuid**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.geteuid.**geteuid32**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.getgid

**class** angr.procedures.linux_kernel.getgid.**getgid**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.getgid.**getgid32**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.getpid

**class** angr.procedures.linux_kernel.getpid.**getpid**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.getpid.**getppid**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.getrlimit

**class** angr.procedures.linux_kernel.getrlimit.**getrlimit**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.getrlimit.**ugetrlimit**

Bases: *getrlimit*

### angr.procedures.linux_kernel.gettid

**class** angr.procedures.linux_kernel.gettid.**gettid**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.getuid

**class** angr.procedures.linux_kernel.getuid.**getuid**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.getuid.**getuid32**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.iovec

**class** angr.procedures.linux_kernel.iovec.**readv**

Bases: *SimProcedure*

**class** angr.procedures.linux_kernel.iovec.**writev**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.lseek

**class** angr.procedures.linux_kernel.lseek.**lseek**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.mmap

**class** angr.procedures.linux_kernel.mmap.**old_mmap**

Bases: *mmap*

**class** angr.procedures.linux_kernel.mmap.**mmap2**

Bases: *mmap*

### angr.procedures.linux_kernel.mprotect

**class** angr.procedures.linux_kernel.mprotect.**mprotect**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.munmap

**class** angr.procedures.linux_kernel.munmap.**munmap**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.openat

class angr.procedures.linux_kernel.openat.**openat**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.set_tid_address

class angr.procedures.linux_kernel.set_tid_address.**set_tid_address**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.sigaction

class angr.procedures.linux_kernel.sigaction.**sigaction**

Bases: *SimProcedure*

class angr.procedures.linux_kernel.sigaction.**rt_sigaction**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.sigprocmask

class angr.procedures.linux_kernel.sigprocmask.**sigprocmask**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.stat

class angr.procedures.linux_kernel.stat.**stat**

Bases: *fstat*

### angr.procedures.linux_kernel.sysinfo

class angr.procedures.linux_kernel.sysinfo.**sysinfo**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.tgkill

class angr.procedures.linux_kernel.tgkill.**tgkill**

Bases: *SimProcedure*

### angr.procedures.linux_kernel.time

class angr.procedures.linux_kernel.time.**time**

Bases: *SimProcedure*

**KEY = 'sys_last_time'**

**property last_time**

### angr.procedures.linux_kernel.uid

class angr.procedures.linux_kernel.uid.**getuid**

Bases: *SimProcedure*

class angr.procedures.linux_kernel.uid.**getgid**

Bases: *SimProcedure*

class angr.procedures.linux_kernel.uid.**getresgid**

Bases: *SimProcedure*

class angr.procedures.linux_kernel.uid.**getresuid**

Bases: *SimProcedure*

#### angr.procedures.linux_kernel.uname

class angr.procedures.linux_kernel.uname.**uname**

Bases: *SimProcedure*

#### angr.procedures.linux_kernel.unlink

class angr.procedures.linux_kernel.unlink.**unlink**

Bases: *SimProcedure*

#### angr.procedures.linux_kernel.vsyscall

### 10.27.14 angr.procedures.linux_loader

These function implement functions or other functionalities provided by the linux userspace dynamic loader

#### Submodules

*sim_loader*
*tls*

#### angr.procedures.linux_loader.sim_loader

class angr.procedures.linux_loader.sim_loader.**LinuxLoader**

Bases: *SimProcedure*

**NO_RET = True**

**local_vars: tuple[str, ...] = ('initializers',)**

**run_initializer**()

class angr.procedures.linux_loader.sim_loader.**IFuncResolver**

Bases: *SimProcedure*

**NO_RET = True**

**local_vars: tuple[str, ...] = ('saved_regs',)**

**after_call**(*funcaddr=None*)

#### angr.procedures.linux_loader.tls

class angr.procedures.linux_loader.tls.**tlsdesc_resolver**

Bases: *SimProcedure*

### 10.27.15 angr.procedures.msvcr

These functions implement those found in the various versions of msvcr.dll (Microsoft Visual C/C++ Runtime)

#### Submodules

| |
|---|
| *fmode* |

#### angr.procedures.msvcr.fmode

### 10.27.16 angr.procedures.ntdll

#### Submodules

| |
|---|
| *exceptions* |

#### angr.procedures.ntdll.exceptions

**class** angr.procedures.ntdll.exceptions.**KiUserExceptionDispatcher**

Bases: *SimProcedure*

**local_vars: tuple[str, ...] = ('tib_ptr', 'top_record', 'cur_ptr')**

**IS_FUNCTION = False**

**tib_ptr = None**

**top_record = None**

**cur_ptr = None**

**dispatch**(*record*, *context*)

### 10.27.17 angr.procedures.posix

These procedures implement functions described in POSIX.1

#### Submodules

| |
|---|
| *accept* |
| *bind* |
| *bzero* |
| *chroot* |
| *close* |
| *closedir* |
| *dup* |
| *fcntl* |
| *fdopen* |
| *fileno* |
| *fork* |
| *getenv* |
| *gethostbyname* |

continues on next page

Table 88 – continued from previous page

| |
|---|
| *getpass* |
| *getsockopt* |
| *htonl* |
| *htons* |
| *inet_ntoa* |
| *listen* |
| *mmap* |
| *open* |
| *opendir* |
| *poll* |
| *pread64* |
| *pthread* |
| *pwrite64* |
| *read* |
| *readdir* |
| *recv* |
| *recvfrom* |
| *select* |
| *send* |
| *setsockopt* |
| *sigaction* |
| *sim_time* |
| *sleep* |
| *socket* |
| *strcasecmp* |
| *strdup* |
| *strncasecmp* |
| *strndup* |
| *strtok_r* |
| *syslog* |
| *tz* |
| *unlink* |
| *usleep* |
| *write* |

### angr.procedures.posix.accept

**class** angr.procedures.posix.accept.**accept**

Bases: *SimProcedure*

### angr.procedures.posix.bind

**class** angr.procedures.posix.bind.**bind**

Bases: *SimProcedure*

### angr.procedures.posix.bzero

**class** angr.procedures.posix.bzero.**bzero**

Bases: *memset*

### angr.procedures.posix.chroot

**class** angr.procedures.posix.chroot.**chroot**

Bases: *SimProcedure*

Changes the root directory of the emulated filesystem (`state.fs`) to the given path.

The new root is resolved inside the filesystem the state already has, so this can only narrow what the guest is able to reach.

### angr.procedures.posix.close

**class** angr.procedures.posix.close.**close**

Bases: *SimProcedure*

### angr.procedures.posix.closedir

**class** angr.procedures.posix.closedir.**closedir**

Bases: *close*

### angr.procedures.posix.dup

**class** angr.procedures.posix.dup.**dup**

Bases: *SimProcedure*

**class** angr.procedures.posix.dup.**dup2**

Bases: *SimProcedure*

**class** angr.procedures.posix.dup.**dup3**

Bases: *SimProcedure*

### angr.procedures.posix.fcntl

**class** angr.procedures.posix.fcntl.**fcntl**

Bases: *SimProcedure*

### angr.procedures.posix.fdopen

angr.procedures.posix.fdopen.**mode_to_flag**(*mode*)

angr.procedures.posix.fdopen.**create_file**(*mode*)

**class** angr.procedures.posix.fdopen.**fdopen**

Bases: *SimProcedure*

### angr.procedures.posix.fileno

**class** angr.procedures.posix.fileno.**fileno**

Bases: *SimProcedure*

angr.procedures.posix.fileno.**fileno_unlocked**

alias of *fileno*

### angr.procedures.posix.fork

class angr.procedures.posix.fork.**fork**

Bases: *SimProcedure*

### angr.procedures.posix.getenv

class angr.procedures.posix.getenv.**getenv**

Bases: *SimProcedure*

The getenv() function searches the environment list to find the environment variable name, and returns a pointer to the corresponding value string.

### angr.procedures.posix.gethostbyname

class angr.procedures.posix.gethostbyname.**gethostbyname**

Bases: *SimProcedure*

### angr.procedures.posix.getpass

class angr.procedures.posix.getpass.**getpass**

Bases: *SimProcedure*

### angr.procedures.posix.getsockopt

class angr.procedures.posix.getsockopt.**getsockopt**

Bases: *SimProcedure*

### angr.procedures.posix.htonl

class angr.procedures.posix.htonl.**htonl**

Bases: *SimProcedure*

### angr.procedures.posix.htons

class angr.procedures.posix.htons.**htons**

Bases: *SimProcedure*

### angr.procedures.posix.inet_ntoa

class angr.procedures.posix.inet_ntoa.**inet_ntoa**

Bases: *SimProcedure*

inet_ntoa simprocedure

**INET_INADDRSTRLEN = 16**

**run**(*addr_in*)

Run the simprocedure

**Parameters**

**addr_in** (*SimStructValue*) – inet_addr struct (which is just a 32-bit int)

### angr.procedures.posix.listen

**class** angr.procedures.posix.listen.**listen**

Bases: *SimProcedure*

### angr.procedures.posix.mmap

**class** angr.procedures.posix.mmap.**mmap**

Bases: *SimProcedure*

**allocate_memory**(*size*)

### angr.procedures.posix.open

**class** angr.procedures.posix.open.**open**

Bases: *SimProcedure*

### angr.procedures.posix.opendir

**class** angr.procedures.posix.opendir.**opendir**

Bases: *SimProcedure*

### angr.procedures.posix.poll

**class** angr.procedures.posix.poll.**poll**

Bases: *SimProcedure*

### angr.procedures.posix.pread64

**class** angr.procedures.posix.pread64.**pread64**

Bases: *SimProcedure*

### angr.procedures.posix.pthread

**class** angr.procedures.posix.pthread.**pthread_create**

Bases: *SimProcedure*

Simulates the new thread as an equally viable branch of symbolic execution.

**ADDS_EXITS = True**

**terminate_thread**(*thread*, *attr*, *start_routine*, *arg*)

**class** angr.procedures.posix.pthread.**pthread_cond_signal**

Bases: *SimProcedure*

A no-op.

**class** angr.procedures.posix.pthread.**pthread_mutex_lock**

Bases: *SimProcedure*

Always returns 0 (SUCCESS).

**class** angr.procedures.posix.pthread.**pthread_mutex_unlock**

Bases: *SimProcedure*

Always returns 0 (SUCCESS).

class angr.procedures.posix.pthread.**pthread_once**

Bases: *SimProcedure*

**retsite**(*control*, *func*)

### angr.procedures.posix.pwrite64

class angr.procedures.posix.pwrite64.**pwrite64**

Bases: *SimProcedure*

### angr.procedures.posix.read

class angr.procedures.posix.read.**read**

Bases: *SimProcedure*

### angr.procedures.posix.readdir

angr.procedures.posix.readdir.**Dirent**

alias of dirent

class angr.procedures.posix.readdir.**readdir**

Bases: *SimProcedure*

**struct = None**

**condition = None**

**instrument**()

Override this function to instrument the SimProcedure.

The two useful variables you can override are self.struct, a named tuple of all the struct fields, and self.condition, the condition for whether the function succeeds.

### angr.procedures.posix.recv

class angr.procedures.posix.recv.**recv**

Bases: *SimProcedure*

### angr.procedures.posix.recvfrom

class angr.procedures.posix.recvfrom.**recvfrom**

Bases: *SimProcedure*

### angr.procedures.posix.select

class angr.procedures.posix.select.**select**

Bases: *SimProcedure*

### angr.procedures.posix.send

class angr.procedures.posix.send.**send**

Bases: *SimProcedure*

#### angr.procedures.posix.setsockopt

class angr.procedures.posix.setsockopt.**setsockopt**

Bases: *SimProcedure*

#### angr.procedures.posix.sigaction

class angr.procedures.posix.sigaction.**sigaction**

Bases: *SimProcedure*

#### angr.procedures.posix.sim_time

class angr.procedures.posix.sim_time.**gettimeofday**

Bases: *SimProcedure*

class angr.procedures.posix.sim_time.**clock_gettime**

Bases: *SimProcedure*

#### angr.procedures.posix.sleep

class angr.procedures.posix.sleep.**sleep**

Bases: *SimProcedure*

#### angr.procedures.posix.socket

class angr.procedures.posix.socket.**socket**

Bases: *SimProcedure*

#### angr.procedures.posix.strcasecmp

class angr.procedures.posix.strcasecmp.**strcasecmp**

Bases: *SimProcedure*

#### angr.procedures.posix.strdup

class angr.procedures.posix.strdup.**strdup**

Bases: *SimProcedure*

#### angr.procedures.posix.strncasecmp

class angr.procedures.posix.strncasecmp.**strncasecmp**

Bases: *SimProcedure*

#### angr.procedures.posix.strndup

class angr.procedures.posix.strndup.**strndup**

Bases: *SimProcedure*

#### angr.procedures.posix.strtok_r

class angr.procedures.posix.strtok_r.**strtok_r**

Bases: *SimProcedure*

#### angr.procedures.posix.syslog

**class** angr.procedures.posix.syslog.**syslog**

Bases: *FormatParser*

#### angr.procedures.posix.tz

**class** angr.procedures.posix.tz.**tzset**

Bases: *SimProcedure*

#### angr.procedures.posix.unlink

**class** angr.procedures.posix.unlink.**unlink**

Bases: *SimProcedure*

#### angr.procedures.posix.usleep

**class** angr.procedures.posix.usleep.**usleep**

Bases: *SimProcedure*

#### angr.procedures.posix.write

**class** angr.procedures.posix.write.**write**

Bases: *SimProcedure*

### 10.27.18 angr.procedures.procedure_dict

### 10.27.19 angr.procedures.stubs

These procedures are various misc stubs useful in analysis

#### Submodules

| |
|---|
| *CallReturn* |
| *NoReturnUnconstrained* |
| *Nop* |
| *PathTerminator* |
| *Redirect* |
| *ReturnChar* |
| *ReturnUnconstrained* |
| *UnresolvableCallTarget* |
| *UnresolvableJumpTarget* |
| *UserHook* |
| *b64_decode* |
| *caller* |
| *crazy_scanf* |
| *format_parser* |
| *syscall_stub* |

#### angr.procedures.stubs.CallReturn

**class** angr.procedures.stubs.CallReturn.**CallReturn**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.stubs.NoReturnUnconstrained

class angr.procedures.stubs.NoReturnUnconstrained.**NoReturnUnconstrained**

Bases: *SimProcedure*

Use in places you would put ReturnUnconstrained as a default action but the function shouldn't actually return.

**NO_RET = True**

### angr.procedures.stubs.Nop

class angr.procedures.stubs.Nop.**Nop**

Bases: *SimProcedure*

### angr.procedures.stubs.PathTerminator

class angr.procedures.stubs.PathTerminator.**PathTerminator**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.stubs.Redirect

class angr.procedures.stubs.Redirect.**Redirect**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.stubs.ReturnChar

class angr.procedures.stubs.ReturnChar.**ReturnChar**

Bases: *SimProcedure*

### angr.procedures.stubs.ReturnUnconstrained

class angr.procedures.stubs.ReturnUnconstrained.**ReturnUnconstrained**

Bases: *SimProcedure*

**ARGS_MISMATCH = True**

### angr.procedures.stubs.UnresolvableCallTarget

class angr.procedures.stubs.UnresolvableCallTarget.**UnresolvableCallTarget**

Bases: *SimProcedure*

**NO_RET = False**

### angr.procedures.stubs.UnresolvableJumpTarget

class angr.procedures.stubs.UnresolvableJumpTarget.**UnresolvableJumpTarget**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.stubs.UserHook

class angr.procedures.stubs.UserHook.**UserHook**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.stubs.b64_decode

class angr.procedures.stubs.b64_decode.**b64_decode**

Bases: *SimProcedure*

### angr.procedures.stubs.caller

class angr.procedures.stubs.caller.**Caller**

Bases: *SimProcedure*

Caller stub. Creates a Ijk_Call exit to the specified function

**after_call**(*target_addr=None*, *target_cc=None*)

### angr.procedures.stubs.crazy_scanf

class angr.procedures.stubs.crazy_scanf.**crazy_scanf**

Bases: *SimProcedure*

### angr.procedures.stubs.format_parser

class angr.procedures.stubs.format_parser.**FormatString**

Bases: object

Describes a format string.

**SCANF_DELIMITERS = [b'\t', b'\n', b'\x0b', b'\r', b' ']**

**__init__**(*parser*, *components*)

Takes a list of components which are either just strings or a FormatSpecifier.

**property state**

**replace**(*va_arg*)

Implement printf - based on the stored format specifier information, format the values from the arg getter function *args* into a string.

**Parameters**

**va_arg** – A function which takes a type and returns the next argument of that type

**Returns**

The result formatted string

**interpret**(*va_arg*, *addr=None*, *simfd=None*)

implement scanf - extract formatted data from memory or a file according to the stored format specifiers and store them into the pointers extracted from *args*.

**Parameters**

- **va_arg** – A function which, given a type, returns the next argument of that type
- **addr** – The address in the memory to extract data from, or…

- **simfd** – A file descriptor to use for reading data from

**Returns**
The number of arguments parsed

class angr.procedures.stubs.format_parser.**FormatSpecifier**

Bases: object

Describes a format specifier within a format string.

**__init__**(*sss*, *length_spec*, *pad_chr*, *ty*)

**Parameters**
**ty** (SimType)

**string**

**ty**

**length_spec**

**pad_chr**

**property signed**

**property size**

**property spec_type**

class angr.procedures.stubs.format_parser.**FormatParser**

Bases: *SimProcedure*

For SimProcedures relying on printf-style format strings.

**ARGS_MISMATCH = True**

**basic_spec = {b'A': double, b'E': double, b'F': double, b'G': double, b'X': unsigned int, b'a': double, b'c': char, b'd': int, b'e': double, b'f': double, b'g': double, b'i': int, b'n': unsigned int*, b'o': unsigned int, b'p': unsigned int*, b's': char*, b'u': unsigned int, b'x': unsigned int}**

**int_sign = {'signed': [b'd', b'i'], 'unsigned': [b'o', b'u', b'x', b'X']}**

**int_len_mod = {b'h': (short, unsigned short), b'hh': (char, char), b'j': (long long, unsigned long long), b'l': (long, unsigned long), b'll': (long long, unsigned long long), b't': (long, long), b'z': (size_t, size_t)}**

**float_spec = [b'e', b'E', b'f', b'F', b'g', b'G', b'a', b'A']**

**float_len_mod: dict[*bytes*, *SimType*] = {b'L': double, b'l': double, b'll': double}**

**other_types = {('string',): <function FormatParser.<lambda>>}**

**flags = ['#', '0', '\\-', ' ', '\\+', "\\'", 'I']**

**extract_components**(*fmt*)

Extract the actual formats from the format string *fmt*.

**Parameters**
**fmt** (list) – A list of format chars.

**Return type**
list[bytes | *FormatSpecifier*]

**Returns**
a FormatString object

**class** angr.procedures.stubs.format_parser.**ScanfFormatParser**

Bases: *FormatParser*

For SimProcedures relying on scanf-style format strings.

**basic_spec = {b'A': float, b'E': float, b'F': float, b'G': float, b'X': unsigned int, b'a': float, b'c': char, b'd': int, b'e': float, b'f': float, b'g': float, b'i': int, b'n': unsigned int*, b'o': unsigned int, b'p': unsigned int*, b's': char*, b'u': unsigned int, b'x': unsigned int}**

**float_len_mod: dict[*bytes*, *SimType*] = {b'L': double, b'l': double, b'll': double}**

### angr.procedures.stubs.syscall_stub

**class** angr.procedures.stubs.syscall_stub.**syscall**

Bases: *SimProcedure*

## 10.27.20 angr.procedures.testing

These procedures are used in the angr test suite. I'm like 90% sure they're unused at this point.

### Submodules

*manyargs*
*retreg*

### angr.procedures.testing.manyargs

**class** angr.procedures.testing.manyargs.**manyargs**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.testing.retreg

**class** angr.procedures.testing.retreg.**retreg**

Bases: *SimProcedure*

## 10.27.21 angr.procedures.tracer

These procedures implement system calls for the cgc DECREE platform, in a specific way for tracing.

### Submodules

*random*
*receive*
*transmit*

#### angr.procedures.tracer.random

**class** angr.procedures.tracer.random.**random**

Bases: *random*

This a passthrough to the CGC version. Removing this requires regenerating angrop caches used by rex and so this is being retained.

#### angr.procedures.tracer.receive

**class** angr.procedures.tracer.receive.**receive**

Bases: *receive*

Receive which fixes the input to file descriptor to 0.

#### angr.procedures.tracer.transmit

**class** angr.procedures.tracer.transmit.**transmit**

Bases: *transmit*

Transmit which fixes the output file descriptor to 1.

### 10.27.22 angr.procedures.uclibc

These procedures implement internal functions in the uClibc libc implementation

### 10.27.23 angr.procedures.win32

These procedures implement functions from the win32 api, notably kernel32.dll

#### Submodules

| |
|---|
| *EncodePointer* |
| *ExitProcess* |
| *GetCommandLine* |
| *GetCurrentProcessId* |
| *GetCurrentThreadId* |
| *GetLastInputInfo* |
| *GetModuleHandle* |
| *GetProcessAffinityMask* |
| *InterlockedExchange* |
| *IsProcessorFeaturePresent* |
| *VirtualAlloc* |
| *VirtualProtect* |
| *critical_section* |
| *dynamic_loading* |
| *file_handles* |
| *gethostbyname* |
| *heap* |
| *is_bad_ptr* |
| *local_storage* |
| *mutex* |
| *sim_time* |
| *system_paths* |

### angr.procedures.win32.EncodePointer

class angr.procedures.win32.EncodePointer.**EncodePointer**

Bases: *SimProcedure*

### angr.procedures.win32.ExitProcess

class angr.procedures.win32.ExitProcess.**ExitProcess**

Bases: *SimProcedure*

**NO_RET = True**

### angr.procedures.win32.GetCommandLine

class angr.procedures.win32.GetCommandLine.**GetCommandLineA**

Bases: *SimProcedure*

class angr.procedures.win32.GetCommandLine.**GetCommandLineW**

Bases: *SimProcedure*

### angr.procedures.win32.GetCurrentProcessId

class angr.procedures.win32.GetCurrentProcessId.**GetCurrentProcessId**

Bases: *SimProcedure*

### angr.procedures.win32.GetCurrentThreadId

class angr.procedures.win32.GetCurrentThreadId.**GetCurrentThreadId**

Bases: *SimProcedure*

### angr.procedures.win32.GetLastInputInfo

class angr.procedures.win32.GetLastInputInfo.**GetLastInputInfo**

Bases: *SimProcedure*

**cbSize = None**

**dwTime = None**

**fill_symbolic**()

**fill_concrete**()

### angr.procedures.win32.GetModuleHandle

class angr.procedures.win32.GetModuleHandle.**GetModuleHandleA**

Bases: *SimProcedure*

**handle**(*module_name*)

class angr.procedures.win32.GetModuleHandle.**GetModuleHandleW**

Bases: *GetModuleHandleA*

### angr.procedures.win32.GetProcessAffinityMask

class angr.procedures.win32.GetProcessAffinityMask.**GetProcessAffinityMask**

Bases: *SimProcedure*

**paffinity_mask = None**

**saffinity_mask = None**

**fill_symbolic**()

**fill_concrete**()

### angr.procedures.win32.InterlockedExchange

class angr.procedures.win32.InterlockedExchange.**InterlockedExchange**

Bases: *SimProcedure*

### angr.procedures.win32.IsProcessorFeaturePresent

class angr.procedures.win32.IsProcessorFeaturePresent.**IsProcessorFeaturePresent**

Bases: *SimProcedure*

### angr.procedures.win32.VirtualAlloc

angr.procedures.win32.VirtualAlloc.**convert_prot**(*prot*)

Convert from a windows memory protection constant to an angr bitmask

angr.procedures.win32.VirtualAlloc.**deconvert_prot**(*prot*)

Convert from a angr bitmask to a windows memory protection constant

class angr.procedures.win32.VirtualAlloc.**VirtualAlloc**

Bases: *SimProcedure*

**allocate_memory**(*size*)

### angr.procedures.win32.VirtualProtect

class angr.procedures.win32.VirtualProtect.**VirtualProtect**

Bases: *SimProcedure*

### angr.procedures.win32.critical_section

class angr.procedures.win32.critical_section.**InitializeCriticalSectionAndSpinCount**

Bases: *SimProcedure*

class angr.procedures.win32.critical_section.**InitializeCriticalSectionEx**

Bases: *SimProcedure*

### angr.procedures.win32.dynamic_loading

class angr.procedures.win32.dynamic_loading.**LoadLibraryA**

Bases: *SimProcedure*

**load**(*lib*)

**register**(*obj*)

class angr.procedures.win32.dynamic_loading.**LoadLibraryExW**

Bases: *LoadLibraryA*

class angr.procedures.win32.dynamic_loading.**GetProcAddress**

Bases: *SimProcedure*

**KEY = 'dynamically_loaded_procedures'**

**property procs**

### angr.procedures.win32.file_handles

class angr.procedures.win32.file_handles.**GetStdHandle**

Bases: *SimProcedure*

class angr.procedures.win32.file_handles.**ReadFile**

Bases: *SimProcedure*

class angr.procedures.win32.file_handles.**WriteFile**

Bases: *SimProcedure*

### angr.procedures.win32.gethostbyname

class angr.procedures.win32.gethostbyname.**gethostbyname**

Bases: *SimProcedure*

### angr.procedures.win32.heap

class angr.procedures.win32.heap.**GetProcessHeap**

Bases: *SimProcedure*

class angr.procedures.win32.heap.**HeapCreate**

Bases: *SimProcedure*

class angr.procedures.win32.heap.**HeapAlloc**

Bases: *SimProcedure*

class angr.procedures.win32.heap.**HeapReAlloc**

Bases: *SimProcedure*

class angr.procedures.win32.heap.**GlobalAlloc**

Bases: *HeapAlloc*

class angr.procedures.win32.heap.**HeapFree**

Bases: *SimProcedure*

### angr.procedures.win32.is_bad_ptr

class angr.procedures.win32.is_bad_ptr.**IsBadReadPtr**

Bases: *SimProcedure*

class angr.procedures.win32.is_bad_ptr.**IsBadWritePtr**

Bases: *SimProcedure*

class angr.procedures.win32.is_bad_ptr.**IsBadCodePtr**

Bases: *SimProcedure*

### angr.procedures.win32.local_storage

angr.procedures.win32.local_storage.**mutate_dict**(*state*, *KEY*)

angr.procedures.win32.local_storage.**has_index**(*state*, *idx*, *KEY*)

**class** angr.procedures.win32.local_storage.**TlsAlloc**

Bases: *SimProcedure*

**KEY = 'win32_tls'**

**class** angr.procedures.win32.local_storage.**TlsSetValue**

Bases: *SimProcedure*

**KEY = 'win32_tls'**

**class** angr.procedures.win32.local_storage.**TlsGetValue**

Bases: *SimProcedure*

**KEY = 'win32_tls'**

**class** angr.procedures.win32.local_storage.**TlsFree**

Bases: *SimProcedure*

**KEY = 'win32_tls'**

**SETTER**

alias of *TlsSetValue*

**class** angr.procedures.win32.local_storage.**FlsAlloc**

Bases: *TlsAlloc*

**KEY = 'win32_fls'**

**class** angr.procedures.win32.local_storage.**FlsGetValue**

Bases: *TlsGetValue*

**KEY = 'win32_fls'**

**class** angr.procedures.win32.local_storage.**FlsSetValue**

Bases: *TlsSetValue*

**KEY = 'win32_fls'**

**class** angr.procedures.win32.local_storage.**FlsFree**

Bases: *TlsFree*

**KEY = 'win32_fls'**

**SETTER**

alias of *FlsSetValue*

### angr.procedures.win32.mutex

**class** angr.procedures.win32.mutex.**CreateMutexA**

Bases: *SimProcedure*

**class** angr.procedures.win32.mutex.**CreateMutexEx**

Bases: *CreateMutexA*

### angr.procedures.win32.sim_time

class angr.procedures.win32.sim_time.**GetSystemTimeAsFileTime**

Bases: *SimProcedure*

**timestamp = None**

**instrument**()

**fill_from_timestamp**(*ts*)

**fill_symbolic**()

class angr.procedures.win32.sim_time.**GetLocalTime**

Bases: *SimProcedure*

**wYear = None**

**wMonth = None**

**wDayOfWeek = None**

**wDay = None**

**wHour = None**

**wMinute = None**

**wSecond = None**

**wMilliseconds = None**

**instrument**()

Override this method, setting the class vars of the same names as the windows SYSTEMTIME structure. By default, fills it with appropriately constrained symbolic data, but makes no attempt to make sure that the value can make sense semantically, e.g. feb 31 can be represented.

...unless the USE_SYSTEM_TIMES state option is present, in which case it will use the system time.

https://msdn.microsoft.com/en-us/library/windows/desktop/ms724950(v=vs.85).aspx

**fill_symbolic**()

Fill the class with constrained symbolic values.

**fill_from_timestamp**(*ts*)

Fill the class with the appropriate values extracted from the given timestamp.

**Parameters**
**ts** – A POSIX timestamp.

class angr.procedures.win32.sim_time.**QueryPerformanceCounter**

Bases: *SimProcedure*

class angr.procedures.win32.sim_time.**GetTickCount**

Bases: *SimProcedure*

class angr.procedures.win32.sim_time.**GetTickCount64**

Bases: *SimProcedure*

**KEY = ('sim_time', 'GetTickCount')**

#### angr.procedures.win32.system_paths

class angr.procedures.win32.system_paths.**GetTempPathA**

Bases: *SimProcedure*

**RESULT = <BV64 0x433a5c54656d705c>**

class angr.procedures.win32.system_paths.**GetWindowsDirectoryA**

Bases: *SimProcedure*

**RESULT = <BV80 0x433a5c57696e646f7773>**

### 10.27.24 angr.procedures.win32_kernel

These procedures implement functions from the Windows kernel.

#### Submodules

*ExAllocatePool*
*ExFreePoolWithTag*

#### angr.procedures.win32_kernel.ExAllocatePool

class angr.procedures.win32_kernel.ExAllocatePool.**ExAllocatePool**

Bases: *SimProcedure*

#### angr.procedures.win32_kernel.ExFreePoolWithTag

class angr.procedures.win32_kernel.ExFreePoolWithTag.**ExFreePoolWithTag**

Bases: *SimProcedure*

### 10.27.25 angr.procedures.win_user32

#### Submodules

*chars*
*keyboard*
*messagebox*

#### angr.procedures.win_user32.chars

class angr.procedures.win_user32.chars.**CharNextA**

Bases: *SimProcedure*

class angr.procedures.win_user32.chars.**CharPrevA**

Bases: *SimProcedure*

#### angr.procedures.win_user32.keyboard

class angr.procedures.win_user32.keyboard.**GetKeyboardType**

Bases: *SimProcedure*

#### angr.procedures.win_user32.messagebox

class angr.procedures.win_user32.messagebox.**MessageBoxA**

Bases: *SimProcedure*

**extract**(*addr*)

class angr.procedures.win_user32.messagebox.**MessageBoxExA**

Bases: *MessageBoxA*

class angr.procedures.win_user32.messagebox.**MessageBoxW**

Bases: *MessageBoxA*

**extract**(*addr*)

class angr.procedures.win_user32.messagebox.**MessageBoxExW**

Bases: *MessageBoxW*, *MessageBoxExA*

class angr.procedures.win_user32.messagebox.**MessageBoxIndirectA**

Bases: *MessageBoxExA*

class angr.procedures.win_user32.messagebox.**MessageBoxIndirectW**

Bases: *MessageBoxW*, *MessageBoxIndirectA*

## 10.28 angr.project

angr.project.**load_shellcode**(*shellcode*, *arch*, *start_offset=0*, *load_address=0*, *thumb=False*, ***kwargs*)

Load a new project based on a snippet of assembly or bytecode.

**Parameters**

- **shellcode** (bytes | str) – The data to load, as either a bytestring of instructions or a string of assembly text
- **arch** – The name of the arch to use, or an archinfo class
- **start_offset** – The offset into the data to start analysis (default 0)
- **load_address** – The address to place the data in memory (default 0)
- **thumb** – Whether this is ARM Thumb shellcode

class angr.project.**Project**

Bases: object

This is the main class of the angr module. It is meant to contain a set of binaries and the relationships between them, and perform analyses on them.

**Parameters**

- **thing** – The path to the main executable object to analyze, or a CLE Loader object.
- **default_analysis_mode** – The mode of analysis to use by default. Defaults to 'symbolic'.
- **ignore_functions** – A list of function names that, when imported from shared libraries, should never be stepped into in analysis (calls will return an unconstrained value).
- **use_sim_procedures** – Whether to replace resolved dependencies for which simprocedures are available with said simprocedures.
- **exclude_sim_procedures_func** – A function that, when passed a function name, returns whether or not to wrap it with a simprocedure.

- **exclude_sim_procedures_list** – A list of functions to *not* wrap with simprocedures.
- **arch** – The target architecture (auto-detected otherwise).
- **simos** – a SimOS class to use for this project.
- **engine** – The SimEngine class to use for this project.
- **translation_cache** (*bool*) – If True, cache translated basic blocks rather than re-translating them.
- **selfmodifying_code** (bool) – Whether we aggressively support self-modifying code. When enabled, emulation will try to read code from the current state instead of the original memory, regardless of the current memory protections.
- **store_function** – A function that defines how the Project should be stored. Default to pickling.
- **load_function** – A function that defines how the Project should be loaded. Default to unpickling.
- **analyses_preset** (*angr.misc.PluginPreset*) – The plugin preset for the analyses provider (i.e. Analyses instance).

Any additional keyword arguments passed will be passed onto `cle.Loader`.

**Variables**

- **analyses** – The available analyses.
- **entry** – The program entrypoint.
- **factory** – Provides access to important analysis elements such as path groups and symbolic execution results.
- **filename** – The filename of the executable.
- **loader** – The program loader.
- **storage** – Dictionary of things that should be loaded/stored with the Project.

**__init__**(*thing*, *default_analysis_mode=None*, *ignore_functions=None*, *use_sim_procedures=True*, *exclude_sim_procedures_func=None*, *exclude_sim_procedures_list=()*, *arch=None*, *simos=None*, *engine=None*, *load_options=None*, *translation_cache=True*, *selfmodifying_code=False*, *support_selfmodifying_code=None*, *store_function=None*, *load_function=None*, *analyses_preset=None*, *eager_ifunc_resolution=None*, *cache_limits=None*, *rustc_version=None*, *rustc_optimization_level=None*, ***kwargs*)

**Parameters**

- **load_options** (*dict[str, Any] | None*)
- **selfmodifying_code** (*bool*)
- **support_selfmodifying_code** (*bool | None*)
- **cache_limits** (*dict[str, int | None] | None*)

**arch:** *Arch*

**property llm_client**

The LLM client for this project. Lazy-initialized from environment variables on first access. Set manually via `project.llm_client = LLMClient(...)` or configure via environment variables `ANGR_LLM_MODEL`, `ANGR_LLM_API_KEY`, `ANGR_LLM_API_BASE`.

**property kb**

**get_kb**(*name*)

**property analyses:** *AnalysesHubWithDefault*

**hook**(*addr*, *hook=None*, *length=0*, *kwargs=None*, *replace=False*)

Hook a section of code with a custom function. This is used internally to provide symbolic summaries of library functions, and can be used to instrument execution or to modify control flow.

When hook is not specified, it returns a function decorator that allows easy hooking. Usage:

```
# Assuming proj is an instance of angr.Project, we will add a custom hook at␣
↪the entry
# point of the project.
@proj.hook(proj.entry)
def my_hook(state):
    print("Welcome to execution!")
```

**Parameters**

- **addr** – The address to hook.
- **hook** – A `angr.project.Hook` describing a procedure to run at the given address. You may also pass in a SimProcedure class or a function directly and it will be wrapped in a Hook object for you.
- **length** – If you provide a function for the hook, this is the number of bytes that will be skipped by executing the hook by default.
- **kwargs** – If you provide a SimProcedure for the hook, these are the keyword arguments that will be passed to the procedure's *run* method eventually.
- **replace** (`bool` | `None`) – Control the behavior on finding that the address is already hooked. If true, silently replace the hook. If false (default), warn and do not replace the hook. If none, warn and replace the hook.

**is_hooked**(*addr*)

Returns True if *addr* is hooked.

**Parameters**
**addr** – An address.

**Return type**
`bool`

**Returns**
True if addr is hooked, False otherwise.

**hooked_by**(*addr*)

Returns the current hook for *addr*.

**Parameters**
**addr** – An address.

**Return type**
*SimProcedure* | `None`

**Returns**
None if the address is not hooked.

**unhook**(*addr*)

Remove a hook.

**Parameters**
**addr** – The address of the hook.

**hook_symbol**(*symbol_name*, *simproc*, *kwargs=None*, *replace=None*)

Resolve a dependency in a binary. Looks up the address of the given symbol, and then hooks that address. If the symbol was not available in the loaded libraries, this address may be provided by the CLE externs object.

Additionally, if instead of a symbol name you provide an address, some secret functionality will kick in and you will probably just hook that address, UNLESS you're on powerpc64 ABIv1 or some yet-unknown scary ABI that has its function pointers point to something other than the actual functions, in which case it'll do the right thing.

**Parameters**

- **symbol_name** – The name of the dependency to resolve.
- **simproc** – The SimProcedure instance (or function) with which to hook the symbol
- **kwargs** – If you provide a SimProcedure for the hook, these are the keyword arguments that will be passed to the procedure's *run* method eventually.
- **replace** (bool | None) – Control the behavior on finding that the address is already hooked. If true, silently replace the hook. If false, warn and do not replace the hook. If none (default), warn and replace the hook.

**Returns**
The address of the new symbol.

**Return type**
int

**symbol_hooked_by**(*symbol_name*)

Return the SimProcedure, if it exists, for the given symbol name.

**Parameters**
**symbol_name** (*str*) – Name of the symbol.

**Return type**
*SimProcedure* | None

**Returns**
None if the address is not hooked.

**is_symbol_hooked**(*symbol_name*)

Check if a symbol is already hooked.

**Parameters**
**symbol_name** (*str*) – Name of the symbol.

**Returns**
True if the symbol can be resolved and is hooked, False otherwise.

**Return type**
bool

**unhook_symbol**(*symbol_name*)

Remove the hook on a symbol. This function will fail if the symbol is provided by the extern object, as that would result in a state where analysis would be unable to cope with a call to this symbol.

**rehook_symbol**(*new_address*, *symbol_name*, *stubs_on_sync*)

Move the hook for a symbol to a specific address :type new_address: :param new_address: the new address that will trigger the SimProc execution :type symbol_name: :param symbol_name: the name of the symbol (f.i. strcmp ) :return: None

**execute**(**args*, ***kwargs*)

This function is a symbolic execution helper in the simple style supported by triton and manticore. It designed to be run after setting up hooks (see Project.hook), in which the symbolic state can be checked.

This function can be run in three different ways:

- When run with no parameters, this function begins symbolic execution from the entrypoint.
- It can also be run with a "state" parameter specifying a SimState to begin symbolic execution from.
- Finally, it can accept any arbitrary keyword arguments, which are all passed to project.factory.full_init_state.

If symbolic execution finishes, this function returns the resulting simulation manager.

**terminate_execution**()

Terminates a symbolic execution that was started with Project.execute().

**pcode_block_lifter**(*arch*)

Get the p-code basic block lifter, and therefore the Sleigh context, this project decodes *arch* with.

Sleigh records context variables per address, so the blocks of one binary decode with one context and no block of another binary touches it. *arch* is normally this project's own; a *Block* may name another, and decoding that with this project's context would answer for the wrong architecture, so it gets its own.

**Return type**
*PcodeBasicBlockLifter*

**Parameters**
**arch** (*Arch*)

**languages**()

**Return type**
list[str]

**property language_confidence: str | None**

How sure LanguageDetector is about *languages()* ("high", "medium" or "low"), or None when the language did not come from detection.

**property language_is_certain: bool**

Whether the detected language is solid enough to base language-specific ABI decisions on. A single matching symbol name is enough for LanguageDetector to report a language, but not enough to reinterpret every argument and return value in the binary.

**property is_rust_binary: bool**

**property is_go_binary: bool**

Whether the main binary was confidently identified as Go. Go's ABI differs from the platform default in every argument and return value, so a low-confidence detection is deliberately not enough.

**get_function_cache_limit**()

Get the cache limit for function-level caches.

**Return type**
int | None

**Returns**
The cache limit, or None for disabling the cache.

`get_cfg_node_cache_limit()`

Get the cache limit for CFG node caches.

**Return type**
`int | None`

**Returns**
The cache limit, or None to disable the cache.

`get_cfg_edge_cache_limit()`

Get the cache limit for CFG edge caches (adjacency data spilling).

**Return type**
`int | None`

**Returns**
The cache limit, or None to disable the cache.

## 10.29 angr.protos

### Submodules

| | |
|---|---|
| *ail_types_pb2* | Generated protocol buffer code. |
| *cfg_pb2* | Generated protocol buffer code. |
| *clinic_pb2* | Generated protocol buffer code. |
| *codegen_pb2* | Generated protocol buffer code. |
| *decompilation_cache_pb2* | Generated protocol buffer code. |
| *function_pb2* | Generated protocol buffer code. |
| *primitives_pb2* | Generated protocol buffer code. |
| *variables_pb2* | Generated protocol buffer code. |
| *xrefs_pb2* | Generated protocol buffer code. |

### 10.29.1 angr.protos.ail_types_pb2

Generated protocol buffer code.

`class angr.protos.ail_types_pb2.AilEdge`

Bases: `Message`, `Message`

`DESCRIPTOR = <google._upb._message.Descriptor object>`

`class angr.protos.ail_types_pb2.AilEdgeData`

Bases: `Message`, `Message`

`DESCRIPTOR = <google._upb._message.Descriptor object>`

`class angr.protos.ail_types_pb2.AilGraph`

Bases: `Message`, `Message`

`DESCRIPTOR = <google._upb._message.Descriptor object>`

`class angr.protos.ail_types_pb2.ArgVVar`

Bases: `Message`, `Message`

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.ArgVVars

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class EntriesEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.FixedBufferMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.FixedBufferPtrMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.IteExpr

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.IteExprs

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.StaticBuffers

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class EntriesEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.StaticVVar

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.ail_types_pb2.StaticVVars

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class EntriesEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

### 10.29.2 angr.protos.cfg_pb2

Generated protocol buffer code.

class angr.protos.cfg_pb2.**BlockIDProto**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.cfg_pb2.**CFG**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.cfg_pb2.**CFGENode**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.cfg_pb2.**CFGEdgeData**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.cfg_pb2.**CFGNode**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.cfg_pb2.**IndirectJump**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.cfg_pb2.**JumptableInfo**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.cfg_pb2.**MemoryData**

Bases: Message, Message

**Alignment = 11**

**CodeReference = 7**

**DESCRIPTOR = <google._upb._message.Descriptor object>**

**EH4ScopeTable = 17**

**EHFuncInfo = 15**

**EHHandlerType = 19**

**EHTryBlockMap = 18**

**EHUnwindMapEntry = 16**

**ELFHeader = 9**

FloatingPoint = 10

GOTPLTEntry = 8

Integer = 2

PEDelayImportDirectory = 14

PEExportDirectory = 13

PEImportDirectory = 12

PointerArray = 3

SegmentBoundary = 6

String = 4

UnicodeString = 5

UnknownDataType = 0

Unspecified = 1

class angr.protos.cfg_pb2.OptionalInt64

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

### 10.29.3 angr.protos.clinic_pb2

Generated protocol buffer code.

class angr.protos.clinic_pb2.AddressPair

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.clinic_pb2.AddressTuple

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.clinic_pb2.Clinic

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class InlinedCountsEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class StackItemsEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class VvarToVvarEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.clinic_pb2.**ExternBytes**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.clinic_pb2.**StackItem**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

### 10.29.4 angr.protos.codegen_pb2

Generated protocol buffer code.

class angr.protos.codegen_pb2.**AddrToLabelEntry**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**AstToPosEntry**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**CAILBlockMsg**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**CAssignmentMsg**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**CBinaryOpMsg**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**CBreakMsg**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**CConstantMsg**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**CConstantReferenceValue**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.**CConstructNode**

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CConstructTags

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class JsonValuesEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CContinueMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CDirtyExpressionMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CDirtyStatementMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CExpressionStatementMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CFakeVariableMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CForLoopMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CFunctionCallConfigEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CFunctionCallMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CFunctionMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CGotoMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CITEMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CIfBreakMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CIfElseBranch

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CIfElseMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CIncompleteSwitchCaseEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CIncompleteSwitchCaseMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CIndexedVariableMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CLabelMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CLoopMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CMultiStatementExpressionMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CRegisterMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CReturnMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CStatementsMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CStructFieldMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CSwitchCaseEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CSwitchCaseMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CTypeCastMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CUnaryOpMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CUnsupportedStatementMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CVEXCCallExpressionMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CVariableFieldMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CVariableMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.CVectorConvertMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.Codegen

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class ExprCommentsEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class NotesJsonEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class StmtCommentsEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.ConstFormatEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class FmtEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.InstructionMappingEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.InstructionMappingMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.NodeConfigEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.NodeConfigMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.PositionMappingEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.PositionMappingMsg

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.VariableInUseEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.codegen_pb2.VlaDimEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

### 10.29.5 angr.protos.decompilation_cache_pb2

Generated protocol buffer code.

class angr.protos.decompilation_cache_pb2.BinopOperatorEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.decompilation_cache_pb2.DecompilationCache

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class NotesJsonEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.decompilation_cache_pb2.DecompilationParameters

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class ExprCommentsEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class StmtCommentsEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.decompilation_cache_pb2.OptionEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.decompilation_cache_pb2.StackvarMaxSizeEntry

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

### 10.29.6 angr.protos.function_pb2

Generated protocol buffer code.

class angr.protos.function_pb2.CallSite

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.function_pb2.**Function**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

### 10.29.7 angr.protos.primitives_pb2

Generated protocol buffer code.

class angr.protos.primitives_pb2.**Block**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.primitives_pb2.**BlockGraph**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.primitives_pb2.**CodeReference**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.primitives_pb2.**Edge**

Bases: Message, Message

**Boring = 1**

**Call = 2**

**ClientReq = 15**

**DESCRIPTOR = <google._upb._message.Descriptor object>**

**EmFail = 18**

**EmWarn = 9**

**Exception = 16**

**FakeReturn = 4**

**FlushDCache = 19**

**InvalICache = 20**

**MapFail = 13**

**NoDecode = 8**

**NoRedir = 14**

**Privileged = 21**

**Return = 3**

**SigBUS = 22**

**SigFPE = 32**

SigFPE_IntDiv = 10

SigFPE_IntOvf = 23

SigILL = 24

SigSEGV = 12

SigTRAP = 11

Sys_int = 33

Sys_int128 = 7

Sys_int129 = 25

Sys_int130 = 26

Sys_int145 = 27

Sys_int210 = 28

Sys_int32 = 29

Sys_syscall = 6

Sys_sysenter = 30

Syscall = 5

UnknownJumpkind = 0

Yield = 31

class angr.protos.primitives_pb2.Endpoint

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.primitives_pb2.ExternalFunction

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.primitives_pb2.ExternalVariable

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

class angr.protos.primitives_pb2.Instruction

Bases: Message, Message

DESCRIPTOR = <google._upb._message.Descriptor object>

### 10.29.8 angr.protos.variables_pb2

Generated protocol buffer code.

class angr.protos.variables_pb2.**ConstantVariable**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**LocalType**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**MemoryVariable**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**Phi2Var**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**RegisterVariable**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**StackVariable**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**TemporaryVariable**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**Var2Unified**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**VariableAccess**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**VariableBase**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**VariableManagerInternal**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

class angr.protos.variables_pb2.**VariableType**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

### 10.29.9 angr.protos.xrefs_pb2

Generated protocol buffer code.

class angr.protos.xrefs_pb2.**XRefs**

Bases: Message, Message

**DESCRIPTOR = <google._upb._message.Descriptor object>**

## 10.30 angr.rust

#### Submodules


*analyses*
*knowledge_plugins*
*mixins*
*optimization_passes*
*sim_type*
*structuring*
*typehoon*
*utils*


### 10.30.1 angr.rust.analyses

#### Submodules


*cleanup_function_identification*
*flirt_sig_propagation*
*rust_calling_convention*
*rust_symbol_recovery*
*rust_type_hints*
*rustc_version_identification*
*struct_builder*
*type_db_loader*


#### angr.rust.analyses.cleanup_function_identification

class angr.rust.analyses.cleanup_function_identification.**CleanupFunctionIdentification**

Bases: *Analysis*

Identify cleanup functions (deallocators, drop glue, etc.) in Rust binaries.

**__init__**()

### angr.rust.analyses.flirt_sig_propagation

**class** angr.rust.analyses.flirt_sig_propagation.**FlirtSigPropagation**

Bases: *Analysis*

Propagate FLIRT signatures through simple wrapper functions.

**__init__**(*cfg*)

### angr.rust.analyses.rust_calling_convention

**class** angr.rust.analyses.rust_calling_convention.**Pathfinder**

Bases: object

Find paths through the CFG that satisfy certain constraints.

**__init__**(*graph*, *srda_mixin=None*)

**Parameters**
**srda_mixin** (SRDAMixin | *None*)

**find_ret2arg0_paths**(*remove_phi=False*)

**static path_to_graph**(*path*)

**find_backward_path**(*block*, *max_length=None*)

**find_forward_path**(*block*, *max_length=None*)

**class** angr.rust.analyses.rust_calling_convention.**RustCallingConventionAnalysis**

Bases: *Analysis*

Infer function prototype including struct return type and struct argument types.

Function prototype is inferred based on collected facts from the callee function body and the caller function body.

**Facts collected from callee:**

1. Memory writes to the first argument
2. Memory reads to all arguments

**Facts collected from caller:**

1. Initialization of function arguments at callsite (memory writes at callsite)
2. Uses of return value after callee is called

**__init__**(*func*, ***, *ail_manager*, *callsite_path=None*, *post_callsite_path=None*, *is_call_expr=None*, *depth=0*, *max_depth=8*, *callsite_discriminant_hint=None*)

**Parameters**
**ail_manager** (*Manager*)

**property prototype**

**class** angr.rust.analyses.rust_calling_convention.**RustCallingConventionModel**

Bases: object

Stores inferred calling convention facts for a Rust function.

**__init__**()

### Submodules


*fact_collector*
*pathfinder*
*rust_calling_convention*
*rust_calling_convention_model*


### angr.rust.analyses.rust_calling_convention.fact_collector

**class** angr.rust.analyses.rust_calling_convention.fact_collector.**MemoryWriteCollector**

Bases: *AILBlockViewer*

Walk ret2arg0 paths and collect direct memory writes (Store statements) to parameter-backed addresses.

**__init__**(*fc*)

**Parameters**
**fc** (FactCollector)

**collect**()

**class** angr.rust.analyses.rust_calling_convention.fact_collector.**CalleeWriteCollector**

Bases: object

Visit all calls in the function graph. When a call forwards the current function's arg0 to a callee, recursively analyze the callee and merge its memory writes into the current function's facts.

**__init__**(*fc*)

**Parameters**
**fc** (FactCollector)

**collect**()

**class** angr.rust.analyses.rust_calling_convention.fact_collector.**ConstRetValueCollector**

Bases: object

Iterate over return blocks and collect constant return values, including values propagated through tail calls.

**__init__**(*fc*)

**Parameters**
**fc** (FactCollector)

**collect**()

**class** angr.rust.analyses.rust_calling_convention.fact_collector.**CallsiteFactCollector**

Bases: object

From the *caller* context (callsite_path), collect memory writes to stack regions that initialize struct arguments passed by reference.

**__init__**(*fc*)

**Parameters**
**fc** (FactCollector)

**collect**()

class angr.rust.analyses.rust_calling_convention.fact_collector.**FactCollector**

Bases: *CFAMixin*, *SRDAMixin*, *DFAMixin*

Orchestrator that runs all sub-collectors and exposes the combined results.

Sub-collectors write back to this object's fields; when collection is done, the results are flushed to the model.

**__init__**(*analysis*)

**collect**()

Run all sub-collectors, then flush results to the model.

### angr.rust.analyses.rust_calling_convention.pathfinder

class angr.rust.analyses.rust_calling_convention.pathfinder.**Pathfinder**

Bases: object

Find paths through the CFG that satisfy certain constraints.

**__init__**(*graph*, *srda_mixin=None*)

**Parameters**
**srda_mixin** (*SRDAMixin | None*)

**find_ret2arg0_paths**(*remove_phi=False*)

**static path_to_graph**(*path*)

**find_backward_path**(*block*, *max_length=None*)

**find_forward_path**(*block*, *max_length=None*)

### angr.rust.analyses.rust_calling_convention.rust_calling_convention

class angr.rust.analyses.rust_calling_convention.rust_calling_convention.
**RustCallingConventionAnalysis**

Bases: *Analysis*

Infer function prototype including struct return type and struct argument types.

Function prototype is inferred based on collected facts from the callee function body and the caller function body.

**Facts collected from callee:**

1. Memory writes to the first argument
2. Memory reads to all arguments

**Facts collected from caller:**

1. Initialization of function arguments at callsite (memory writes at callsite)
2. Uses of return value after callee is called

**__init__**(*func*, ***, *ail_manager*, *callsite_path=None*, *post_callsite_path=None*, *is_call_expr=None*, *depth=0*, *max_depth=8*, *callsite_discriminant_hint=None*)

**Parameters**
**ail_manager** (*Manager*)

**property prototype**

### angr.rust.analyses.rust_calling_convention.rust_calling_convention_model

**class** angr.rust.analyses.rust_calling_convention.rust_calling_convention_model.**RustCallingConventionModel**

Bases: object

Stores inferred calling convention facts for a Rust function.

**__init__**()

### angr.rust.analyses.rust_symbol_recovery

**class** angr.rust.analyses.rust_symbol_recovery.**RustSymbolRecovery**

Bases: *Analysis*

This analysis recovers Rust symbols by analyzing the binary's symbol table and debug information. It identifies Rust-specific symbols, such as those related to traits, generics, and async functions, and annotates them for further analysis.

**OPT_LEVELS = ['0', '1', '2', '3']**

**__init__**(*sig_dirs=None*)

### angr.rust.analyses.rust_type_hints

**class** angr.rust.analyses.rust_type_hints.**RustTypeHintsAnalysis**

Bases: *Analysis*

Collect type hints from Rust-specific patterns in the AIL graph.

**__init__**(*func*, *graph*, *variable_map=None*)

### angr.rust.analyses.rustc_version_identification

**class** angr.rust.analyses.rustc_version_identification.**RustcVersionIdentification**

Bases: *Analysis*

Identify the rustc version used to compile a Rust binary by matching FLIRT signatures across known rustc versions and selecting the version with the highest match count.

**OPT_LEVELS = ['0', '1', '2', '3']**

**__init__**(*sig_dirs=None*)

### angr.rust.analyses.struct_builder

**class** angr.rust.analyses.struct_builder.**StructBuilder**

Bases: *Analysis*

Build Rust struct types from memory write patterns.

**__init__**(*context*, *strict=False*)

**Parameters**
**context** (OptimizationPass)

**build**(*field_exprs*, *struct_ty*)

**Return type**
*Struct* | *Array* | None

#### angr.rust.analyses.type_db_loader

class angr.rust.analyses.type_db_loader.**TypeDBLoader**

Bases: *Analysis*

Load Rust type definitions from a version-specific JSON type database.

**__init__**()

### 10.30.2 angr.rust.knowledge_plugins

class angr.rust.knowledge_plugins.**ClinicFactory**

Bases: *KnowledgeBasePlugin*

Cache and provide Clinic analysis results for Rust functions.

**__init__**(*kb*)

**get**(*func*, *optimization_passes=None*, *end_stage=None*)

class angr.rust.knowledge_plugins.**KnownStructs**

Bases: *KnowledgeBasePlugin*

Store known Rust struct type definitions.

**__init__**(*kb*)

**get**(*item*, *default=None*)

**match_with_known_structs**(*fields*)

**Return type**
*RustSimStruct* | None

class angr.rust.knowledge_plugins.**Librust**

Bases: *KnowledgeBasePlugin*, *SimLibrary*

Rust standard library procedure definitions.

**__init__**(*kb*)

class angr.rust.knowledge_plugins.**RustCallingConventions**

Bases: *KnowledgeBasePlugin*, MutableMapping

Store inferred Rust calling conventions keyed by function address.

**__init__**(*kb*)

**copy**()

class angr.rust.knowledge_plugins.**TypeHints**

Bases: *KnowledgeBasePlugin*

Store type hints for virtual variables inferred from Rust patterns.

**__init__**(*kb*)

**add_type_hint**(*vvar*, *ty*, *func_addr*)

**Return type**
None

**Parameters**
**func_addr** (*int*)

`get_type_hints`(*func_addr*)

**Return type**
dict[int, *TypeConstant*]

**Parameters**
**func_addr** (*int*)

`copy`()

**Return type**
*TypeHints*

## Submodules


*clinic_factory*
*known_structs*
*librust*
*rust_calling_conventions*
*type_hints*


### angr.rust.knowledge_plugins.clinic_factory

`class` angr.rust.knowledge_plugins.clinic_factory.**ClinicFactory**

Bases: *KnowledgeBasePlugin*

Cache and provide Clinic analysis results for Rust functions.

`__init__`(*kb*)

`get`(*func*, *optimization_passes=None*, *end_stage=None*)

### angr.rust.knowledge_plugins.known_structs

`class` angr.rust.knowledge_plugins.known_structs.**StructMatcher**

Bases: object

Match memory write patterns against known Rust struct layouts.

`__init__`(*project*)

`match`(*fields*)

`class` angr.rust.knowledge_plugins.known_structs.**KnownStructs**

Bases: *KnowledgeBasePlugin*

Store known Rust struct type definitions.

`__init__`(*kb*)

`get`(*item*, *default=None*)

`match_with_known_structs`(*fields*)

**Return type**
*RustSimStruct* | None

#### angr.rust.knowledge_plugins.librust

class angr.rust.knowledge_plugins.librust.**Librust**

Bases: *KnowledgeBasePlugin*, *SimLibrary*

Rust standard library procedure definitions.

**__init__**(*kb*)

#### angr.rust.knowledge_plugins.rust_calling_conventions

class angr.rust.knowledge_plugins.rust_calling_conventions.**RustCallingConventions**

Bases: *KnowledgeBasePlugin*, MutableMapping

Store inferred Rust calling conventions keyed by function address.

**__init__**(*kb*)

**copy**()

#### angr.rust.knowledge_plugins.type_hints

class angr.rust.knowledge_plugins.type_hints.**TypeHints**

Bases: *KnowledgeBasePlugin*

Store type hints for virtual variables inferred from Rust patterns.

**__init__**(*kb*)

**add_type_hint**(*vvar*, *ty*, *func_addr*)

**Return type**
None

**Parameters**
**func_addr** (*int*)

**get_type_hints**(*func_addr*)

**Return type**
dict[int, *TypeConstant*]

**Parameters**
**func_addr** (*int*)

**copy**()

**Return type**
*TypeHints*

### 10.30.3 angr.rust.mixins

class angr.rust.mixins.**CFAMixin**

Bases: object

Control Flow Analysis Mixin

**__init__**(*graph*, *project*)

**num_predecessors**(*block*)

**get_one_predecessor**(*block*)

**Return type**
Block

**num_successors**(*block*)

**get_one_successor**(*block*)

**Return type**
Block

**static first_non_label_stmt**(*block*)

**Return type**
*Statement* | None

**static last_stmt**(*block*)

**Return type**
*Statement* | None

**static replace_stmt**(*block*, *stmts*, *replacement*)

**terminal_call**(*block*)

**Return type**
*Call* | None

**get_call_target**(*call*)

**Return type**
str | None

**Parameters**
**call** (Call)

**match_call**(*block_or_stmt*, *expected*, *monopolize=True*, *use_trait_name=True*)

**class** angr.rust.mixins.**CFGTransformationMixin**

Bases: object

Mixin providing CFG graph transformation operations (block removal, splitting, etc.).

**__init__**(*graph*)

**update_block_indexes**()

**remove_jump_target**(*block*, *jump_target*, *jump_target_idx*)

**Parameters**

- **block** (*Block*)
- **jump_target** (Const | *int*)
- **jump_target_idx** (*int* | *None*)

**remove_false_branch**(*block*)

**Parameters**
**block** (*Block*)

**replace_jump_target**(*block*, *old_target*, *old_target_idx*, *new_target*, *new_target_idx*)

**Parameters**

- **old_target** (*Const | int | None*)
- **old_target_idx** (*int | None*)
- **new_target** (*Const | int*)
- **new_target_idx** (*int | None*)

**remove_block**(*block*)

**Parameters**
**block** (*Block*)

**split_block**(*block*, *new_head_stmt*)

**Parameters**
**block** (*Block*)

**class** angr.rust.mixins.**DFAMixin**

Bases: object

Data Flow Analysis Helper

**__init__**(*graph=None*)

**collect_stack_defs_at**(*block*)

Collect stack definitions at specific block

**collect_callsite_stack_defs**(*callsite_block*, *max_blocks=1*)

Collect stack variable definitions at a given callsite

**static extract_write_to_stack_vvar**(*stmt*)

**Return type**
tuple[*VirtualVariable* | None, *Expression* | None]

**static extract_stack_data_flow**(*stmt*)

**find_reg_ptr_to_reg_data_flow**(*block*, *reg_vvar*)

**find_stack_data_flow**(*block*, *src_offset*, *size*)

**static extract_stack_to_reg_data_flow**(*stmt*)

**find_stack_to_reg_data_flow**(*block*, *src_offset*, *size*)

**get_def_block_and_stmt**(*data*)

**class** angr.rust.mixins.**SRDAMixin**

Bases: object

Mixin providing SSA reaching definitions analysis access.

**__init__**(*subject*, *graph*, *project*, *variable_map*)

**Parameters**
**variable_map** (VariableMap)

**get_vvar_value**(*vvar*)

**Return type**
*Expression* | None

**Parameters**
**vvar** (VirtualVariable)

**get_terminal_vvar_values**(*vvar*, *visited=None*)

**get_terminal_vvar_value**(*vvar*, *visited=None*)

**get_terminal_vvar**(*vvar*, *visited=None*)

**get_stack_vvar_by_insn**(*stack_offset*, *addr*, *block_idx=None*, *size=None*, *op_type=ObservationPointType.OP_BEFORE*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **stack_offset** (*int*)
- **addr** (*int*)
- **block_idx** (*int | None*)

**get_stack_vvar_and_offset_by_insn**(*stack_offset*, *addr*, *block_idx=None*, *op_type=ObservationPointType.OP_BEFORE*)

**Return type**
tuple[*VirtualVariable*, int] | tuple[None, None]

**Parameters**

- **stack_offset** (*int*)
- **addr** (*int*)
- **block_idx** (*int | None*)

**get_vvar_type**(*vvar*)

**Return type**
*RustSimType* | None

**get_def_by_vvar**(*vvar*)

**class** angr.rust.mixins.**SSAVariableMixin**

Bases: object

Mixin for creating and fixing SSA stack virtual variables.

**__init__**(*context*)

**Parameters**
**context** (OptimizationPass)

**new_stack_vvar**(*dst_offset*, *bits*, *tags*, *record=True*)

**fix_stack_vvar_uses**()

### Submodules


*cfa_mixin*
*cfg_transformation_mixin*
*dfa_mixin*
*srda_mixin*
*ssa_variable_mixin*


### angr.rust.mixins.cfa_mixin

**class** angr.rust.mixins.cfa_mixin.**CFAMixin**

Bases: object

Control Flow Analysis Mixin

**__init__**(*graph*, *project*)

**num_predecessors**(*block*)

**get_one_predecessor**(*block*)

**Return type**
Block

**num_successors**(*block*)

**get_one_successor**(*block*)

**Return type**
Block

**static first_non_label_stmt**(*block*)

**Return type**
*Statement* | None

**static last_stmt**(*block*)

**Return type**
*Statement* | None

**static replace_stmt**(*block*, *stmts*, *replacement*)

**terminal_call**(*block*)

**Return type**
*Call* | None

**get_call_target**(*call*)

**Return type**
str | None

**Parameters**
**call** (Call)

**match_call**(*block_or_stmt*, *expected*, *monopolize=True*, *use_trait_name=True*)

### angr.rust.mixins.cfg_transformation_mixin

class angr.rust.mixins.cfg_transformation_mixin.**CFGTransformationMixin**

Bases: object

Mixin providing CFG graph transformation operations (block removal, splitting, etc.).

**__init__**(*graph*)

**update_block_indexes**()

**remove_jump_target**(*block*, *jump_target*, *jump_target_idx*)

**Parameters**

- **block** (*Block*)
- **jump_target** (Const | *int*)
- **jump_target_idx** (*int* | *None*)

**remove_false_branch**(*block*)

**Parameters**
**block** (*Block*)

**replace_jump_target**(*block*, *old_target*, *old_target_idx*, *new_target*, *new_target_idx*)

**Parameters**

- **old_target** (Const | *int* | *None*)
- **old_target_idx** (*int* | *None*)
- **new_target** (Const | *int*)
- **new_target_idx** (*int* | *None*)

**remove_block**(*block*)

**Parameters**
**block** (*Block*)

**split_block**(*block*, *new_head_stmt*)

**Parameters**
**block** (*Block*)

### angr.rust.mixins.dfa_mixin

class angr.rust.mixins.dfa_mixin.**StackDefinition**

Bases: object

A definition of a stack variable.

**data:** *Expression*

**stmt:** *Statement*

**stmt_idx:** *int*

**block:** Block

**__init__**(*data*, *stmt*, *stmt_idx*, *block*)

**Parameters**

- **data** (Expression)
- **stmt** (Statement)
- **stmt_idx** (*int*)
- **block** (*Block*)

**Return type**
None

**class** angr.rust.mixins.dfa_mixin.**DFAMixin**

Bases: object

Data Flow Analysis Helper

**__init__**(*graph=None*)

**collect_stack_defs_at**(*block*)

Collect stack definitions at specific block

**collect_callsite_stack_defs**(*callsite_block*, *max_blocks=1*)

Collect stack variable definitions at a given callsite

**static extract_write_to_stack_vvar**(*stmt*)

**Return type**
tuple[*VirtualVariable* | None, *Expression* | None]

**static extract_stack_data_flow**(*stmt*)

**find_reg_ptr_to_reg_data_flow**(*block*, *reg_vvar*)

**find_stack_data_flow**(*block*, *src_offset*, *size*)

**static extract_stack_to_reg_data_flow**(*stmt*)

**find_stack_to_reg_data_flow**(*block*, *src_offset*, *size*)

**get_def_block_and_stmt**(*data*)

## angr.rust.mixins.srda_mixin

**class** angr.rust.mixins.srda_mixin.**SRDAMixin**

Bases: object

Mixin providing SSA reaching definitions analysis access.

**__init__**(*subject*, *graph*, *project*, *variable_map*)

**Parameters**
**variable_map** (VariableMap)

**get_vvar_value**(*vvar*)

**Return type**
*Expression* | None

**Parameters**
**vvar** (VirtualVariable)

**get_terminal_vvar_values**(*vvar*, *visited=None*)

**get_terminal_vvar_value**(*vvar*, *visited=None*)

**get_terminal_vvar**(*vvar*, *visited=None*)

**get_stack_vvar_by_insn**(*stack_offset*, *addr*, *block_idx=None*, *size=None*, *op_type=ObservationPointType.OP_BEFORE*)

**Return type**
*VirtualVariable* | None

**Parameters**

- **stack_offset** (*int*)
- **addr** (*int*)
- **block_idx** (*int | None*)

**get_stack_vvar_and_offset_by_insn**(*stack_offset*, *addr*, *block_idx=None*, *op_type=ObservationPointType.OP_BEFORE*)

**Return type**
tuple[*VirtualVariable*, int] | tuple[None, None]

**Parameters**

- **stack_offset** (*int*)
- **addr** (*int*)
- **block_idx** (*int | None*)

**get_vvar_type**(*vvar*)

**Return type**
*RustSimType* | None

**get_def_by_vvar**(*vvar*)

### angr.rust.mixins.ssa_variable_mixin

**class** angr.rust.mixins.ssa_variable_mixin.**SSAVariableMixin**

Bases: object

Mixin for creating and fixing SSA stack virtual variables.

**__init__**(*context*)

**Parameters**
**context** (OptimizationPass)

**new_stack_vvar**(*dst_offset*, *bits*, *tags*, *record=True*)

**fix_stack_vvar_uses**()

### 10.30.4 angr.rust.optimization_passes

angr.rust.optimization_passes.**get_rust_optimization_passes**()

#### Submodules


*cleanup_code_remover*
*combo_register_rewriter*
*deref_coercion_simplifier*
*deref_coercion_simplifier_uninlined*
*error_propagation_simplifier*
*function_prototype_inference*
*macro*
*outliners*
*pattern_match_simplifier*
*pre_pattern_match_simplifier*
*redundant_block_remover*
*ret_expr_rewriter*
*rust_calling_convention*
*security_check_remover*
*str_argument_simplifier*
*struct_instantiation_simplifier*
*struct_return_simplifier*
*utils*


#### angr.rust.optimization_passes.cleanup_code_remover

**class** angr.rust.optimization_passes.cleanup_code_remover.**CleanupCodeRemover**

Bases: *OptimizationPass*, *CFGTransformationMixin*, *CFAMixin*, *SRDAMixin*

Remove cleanup code such as deallocation and drop calls.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Remove cleanup code'**

**__init__**(*func*, *manager*, ***kwargs*)

**Parameters**
**manager** (*Manager*)

#### angr.rust.optimization_passes.combo_register_rewriter

**class** angr.rust.optimization_passes.combo_register_rewriter.**ComboRegisterRewriter**

Bases: *OptimizationPass*, *SRDAMixin*

Rewrite return expressions for functions returning struct via multiple registers.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Rewrite return expressions for functions returning struct via multiple registers'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.deref_coercion_simplifier

**class** angr.rust.optimization_passes.deref_coercion_simplifier.**StrCmpSimplifierWalker**

Bases: *AILBlockRewriter*

Rewrite string comparison expressions to use simplified binary ops.

**__init__**(*context*)

**Parameters**
**context** (DerefCoercionSimplifier)

**class** angr.rust.optimization_passes.deref_coercion_simplifier.**DerefCoercionSimplifier**

Bases: *OptimizationPass*, *SRDAMixin*, *CFAMixin*

Simplify explicit deref coercion operations on string arguments.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Simplify explict deref coercion operations'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.deref_coercion_simplifier_uninlined

**class** angr.rust.optimization_passes.deref_coercion_simplifier_uninlined.**DerefCoercionSimplifierUninlined**

Bases: *OptimizationPass*, *SRDAMixin*, *CFAMixin*, *AILBlockRewriter*

Simplify explicit deref coercion operations that have not been inlined. 1. Identify assignments where a VirtualVariable is assigned the result of a deref coercion function call. 2. Record the mapping from the VirtualVariable to the argument of the deref coercion function. 3. Traverse the AIL graph and replace occurrences of the VirtualVariable with the recorded argument. 4. Remove the original assignment statements that performed the deref coercion. 5. Update the AIL graph to reflect these changes.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 8**

**NAME = 'Simplify explicit deref coercion operations (uninlined)'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.error_propagation_simplifier

**class** angr.rust.optimization_passes.error_propagation_simplifier.**ErrorPropagationWalker**

Bases: *SequenceWalker*

Walk sequence nodes to detect and simplify error propagation patterns.

**__init__**(*context*)

**Parameters**
**context** (ErrorPropagationSimplifier)

**class** angr.rust.optimization_passes.error_propagation_simplifier.**ErrorPropagationSimplifier**

Bases: *SequenceOptimizationPass*

Recover the Rust error propagation '?' operator in decompiled output.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 11**

**NAME = 'Recover error propagation "?" operator'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.function_prototype_inference

**class** angr.rust.optimization_passes.function_prototype_inference.**FunctionPrototypeInference**

Bases: *OptimizationPass*, *CFAMixin*, *SSAVariableMixin*

Infer potential struct/enum argument types and return types for function calls.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Infer potential struct/enum argument types and return types'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.macro

### Submodules

*format_macro_simplifier*

### angr.rust.optimization_passes.macro.format_macro_simplifier

**class** angr.rust.optimization_passes.macro.format_macro_simplifier.**FormatWrapperIdentification**

Bases: *Analysis*

Identify functions that wrap format/print operations.

__init__()

property format_wrappers

class angr.rust.optimization_passes.macro.format_macro_simplifier.**FormatWrappers**

Bases: *KnowledgeBasePlugin*

Knowledge base plugin for resolving format wrapper functions.

__init__(*kb*)

resolve(*addr*)

class angr.rust.optimization_passes.macro.format_macro_simplifier.**FormatMacroSimplifier**

Bases: *OptimizationPass*, *CFAMixin*, *DFAMixin*, *SRDAMixin*, *SSAVariableMixin*

Recover print-like macros from format argument construction patterns.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 7

NAME = 'Recover print-like macros'

__init__(*func*, *manager*, ***kwargs*)

replace_call(*call*, *_block*, *_stmt*)

**Parameters**

- **call** (*Call*)
- **_block** (*Block*)

## angr.rust.optimization_passes.outliners

### Submodules

| |
|---|
| *string_cmp_outliner* |
| *string_literal_outliner* |
| *string_outliner* |
| *unwrap_outliner* |
| *vec_outliner* |

### angr.rust.optimization_passes.outliners.string_cmp_outliner

class angr.rust.optimization_passes.outliners.string_cmp_outliner.**StringCmpOutliner**

Bases: *OptimizationPass*

Outline string comparison operations into simplified expressions.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 7

NAME = 'Outline string comparisons'

__init__(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.outliners.string_literal_outliner

class angr.rust.optimization_passes.outliners.string_literal_outliner.**StringLiteralOutliner**

Bases: *OptimizationPass*, *DFAMixin*, *SSAVariableMixin*

Replace string literal construction patterns with StringLiteral expressions.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 7

NAME = 'Outline struct fields to string literals'

__init__(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.outliners.string_outliner

class angr.rust.optimization_passes.outliners.string_outliner.**StringOutliner**

Bases: *OptimizationPass*

Outline String::from and to_string patterns into StringLiteral expressions.

ARCHES = None

PLATFORMS = None

STAGE: *OptimizationPassStage* = 7

NAME = 'Outline String structs to `String::new`'

__init__(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.outliners.unwrap_outliner

class angr.rust.optimization_passes.outliners.unwrap_outliner.**UnwrapSimplifierState**

Bases: object

State for tracking unwrap pattern detection.

__init__(*conditional_jump_block*, *unwrap_failed_block*, *ownership_move_block*, *cmp_expr*, *unwrap_failed_func_name*)

class angr.rust.optimization_passes.outliners.unwrap_outliner.**UnwrapOutliner**

Bases: *OptimizationPass*, *CFAMixin*, *SRDAMixin*, *DFAMixin*, *CFGTransformationMixin*, *SSAVariableMixin*

Replace unwrap/expect patterns with simplified function call expressions.

ARCHES = None

PLATFORMS = None

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Outline unwrap function calls'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.outliners.vec_outliner

**class** angr.rust.optimization_passes.outliners.vec_outliner.**VecOutliner**

Bases: *OptimizationPass*

Replace Vec construction patterns with vec! macro expressions.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Outline Vec structs to `Vec::new`'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.pattern_match_simplifier

**class** angr.rust.optimization_passes.pattern_match_simplifier.**PatternMatchWalker**

Bases: *SequenceWalker*, *DFAMixin*

Walker that converts condition nodes into pattern match or if-let nodes.

**__init__**(*var_manager*, *graph*, *variable_map*)

**class** angr.rust.optimization_passes.pattern_match_simplifier.**PatternMatchSimplifier**

Bases: *SequenceOptimizationPass*

Recover idiomatic Rust pattern match constructs from condition nodes.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 11**

**NAME = 'Recover idiomatic Rust error handling code'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.pre_pattern_match_simplifier

**class**
angr.rust.optimization_passes.pre_pattern_match_simplifier.**PrePatternMatchSimplifier**

Bases: *OptimizationPass*, *ReturnDuplicatorBase*, *DFAMixin*

Duplicate return blocks for identified pattern matches to form if-else structures. For example the following code,

``` if (...){

...

} **else {**
v6 = std::fs::File::open(a1, a2); if !v6 as i32 {
```

```
        ...
    }
} return Err(struct8 {
    field_0: v11
```

**should be converted to**

```
``` v6 = std::fs::File::open(a1, a2); if v6 as i32 {
    return Err(struct8 {
        field_0: v11
    });
} else {
    ...
```

for recovering pattern match constructs in later stage

```
ARCHES = None

PLATFORMS = None

STAGE: OptimizationPassStage = 7

NAME = 'Duplicate return blocks to prepare for pattern match simplification'

DESCRIPTION = '\n Duplicate return blocks for identified pattern matches to form
if-else structures.\n For example the following code,\n ```\n if (...){\n ...\n }
else {\n v6 = std::fs::File::open(a1, a2);\n if !v6 as i32 {\n ...\n }\n }\n return
Err(struct8 {\n field_0:  v11\n });\n ```\n should be converted to\n ```\n v6 =
std::fs::File::open(a1, a2);\n if v6 as i32 {\n return Err(struct8 {\n field_0:
v11\n });\n } else {\n ...\n }\n ```\n for recovering pattern match constructs in
later stage\n '

STRUCTURING: list[str] | None = ['sailr', 'dream']
```

**__init__**(*func*, *manager*, *\**, *vvar_id_start*, *max_calls_in_regions=2*, *minimize_copies_for_regions=True*, *scratch=None*, *\*\*kwargs*)

**Parameters**

- **vvar_id_start** (*int*)
- **max_calls_in_regions** (*int*)
- **minimize_copies_for_regions** (*bool*)
- **scratch** (*dict[str, Any] | None*)

**static extract_scrutinee_and_discriminant**(*condition*)

**static inverse_variant**(*enum_type*, *discriminant*)

**Return type**

*EnumVariant* | None
```

### angr.rust.optimization_passes.redundant_block_remover

class angr.rust.optimization_passes.redundant_block_remover.**RedundantBlockRemover**

Bases: *OptimizationPass*, *CFAMixin*, *CFGTransformationMixin*

Remove redundant blocks that have no effect on functionality.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 9**

**NAME = 'Remove redundant blocks that have no effect on functionality'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.ret_expr_rewriter

class angr.rust.optimization_passes.ret_expr_rewriter.**RetExprRewriter**

Bases: *OptimizationPass*

Rewrite return expressions for functions returning struct via multiple registers.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 1**

**NAME = 'Rewrite return expressions for functions returning struct via multiple registers'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.rust_calling_convention

class angr.rust.optimization_passes.rust_calling_convention.**RustCallingConvention**

Bases: *OptimizationPass*

Apply Rust calling convention analysis results to function prototypes.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 1**

**NAME = 'Recover Rust prototypes and calling convention for current function'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.security_check_remover

class angr.rust.optimization_passes.security_check_remover.**SecurityCheckRemover**

Bases: *OptimizationPass*, *CFAMixin*, *CFGTransformationMixin*

Remove Rust bounds-check and overflow-check panic branches.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Remove security check'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.str_argument_simplifier

**class** angr.rust.optimization_passes.str_argument_simplifier.**StrArgumentSimplifier**

Bases: *OptimizationPass*, *SRDAMixin*

Simplify string literals used as function call arguments in Rust binaries.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 8**

**NAME = 'Simplify string literals used as function call arguments'**

**__init__**(*func*, *manager*, ***kwargs*)

**try_str_literal**(*arg0*, *arg1*)

**try_str_reference**(*arg0*, *arg1*)

Try to identify a &str reference from two arguments. For example, given the following call: Call (

target: 0x4696a0<64>, prototype: ..., args: [

(Reference vvar_400{stack -848}), Load(addr=(Reference vvar_300{combo_reg (16, 32)}), size=8, endness=Iend_LE), Load(addr=((Reference vvar_300{combo_reg (16, 32)}) + 0x8<64>), size=8, endness=Iend_LE), Load(addr=(Reference vvar_307{combo_reg (16, 32)}), size=8, endness=Iend_LE), Load(addr=((Reference vvar_307{combo_reg (16, 32)}) + 0x8<64>), size=8, endness=Iend_LE)

]

) We can identify that the second and third arguments form a &str reference.

**replace_call**(*call*, *_block*, *_stmt*)

**Parameters**

- **call** (Call)
- **_block** (*Block*)

### angr.rust.optimization_passes.struct_instantiation_simplifier

**class** angr.rust.optimization_passes.struct_instantiation_simplifier.**StructBuilder**

Bases: object

Build Struct expressions from sequential memory write patterns.

**__init__**(*context*)

**Parameters**

**context** (StructInstantiationSimplifier)

**build**(*field_exprs*, *struct_ty*)

**Return type**
*Struct* | *Array* | None

**class** angr.rust.optimization_passes.struct_instantiation_simplifier.**StructInstantiationSimplifier**

Bases: *OptimizationPass*, *SRDAMixin*, *CFAMixin*, *DFAMixin*, *SSAVariableMixin*

Simplify struct instantiation from sequential stack writes into Struct expressions.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Make callsite based on known/recovered prototypes'**

**__init__**(*func*, *manager*, ***kwargs*)

### angr.rust.optimization_passes.struct_return_simplifier

**class** angr.rust.optimization_passes.struct_return_simplifier.**StructReturnSimplifier**

Bases: *OptimizationPass*, *SRDAMixin*, *CFGTransformationMixin*

Simplify struct return patterns by wrapping return values in enum constructors.

**ARCHES = None**

**PLATFORMS = None**

**STAGE: *OptimizationPassStage* = 7**

**NAME = 'Simplify function return sites'**

**__init__**(*func*, *manager*, ***kwargs*)

**get_existing_vvar**(*fields*, *block*)

**try_convert_to_enum**(*struct*)

**Parameters**
**struct** (Struct)

**collect_ret_expr**(*path*)

### angr.rust.optimization_passes.utils

angr.rust.optimization_passes.utils.**extract_callee**(*obj*, *kb*)

angr.rust.optimization_passes.utils.**extract_str**(*project*, *str_ptr*, *str_len*)

Extract Rust string literal with given ptr and len

angr.rust.optimization_passes.utils.**extract_str_from_addr**(*project*, *addr*)

**class** angr.rust.optimization_passes.utils.**SideEffectStatementRewriter**

Bases: *AILBlockRewriter*

Rewrite SideEffectStatement nodes via a callback.

**__init__**(*callback*)

**class** angr.rust.optimization_passes.utils.**CallRewriter**

Bases: *AILBlockRewriter*

Rewrite Call expressions and SideEffectStatements via a callback.

**__init__**(*callback*)

angr.rust.optimization_passes.utils.**replace_argument_pairs**(*call*, *callback*)

**Return type**
*Call*

**Parameters**
**call** (*Call*)

### 10.30.5 angr.rust.sim_type

angr.rust.sim_type.**is_composite_type**(*ty*)

**class** angr.rust.sim_type.**RustSimType**

Bases: *SimType*

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*, ***kwargs*)

**Parameters**

- **indent** (*int | None*)
- **name_parens** (*bool*)

**class** angr.rust.sim_type.**RustSimTypeInt**

Bases: *RustSimType*, *SimTypeInt*

**copy**()

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

**property alignment**

The alignment of the type in bytes.

**property size: int**

The size of the type in bits, or None if no size is computable.

**class** angr.rust.sim_type.**RustSimTypeSize**

Bases: *RustSimTypeInt*

**property size**

The size of the type in bits, or None if no size is computable.

**copy**()

class angr.rust.sim_type.**RustSimTypeFunction**

Bases: *RustSimType*, *SimTypeFunction*

SimTypeFunction is a type that specifies an actual function (i.e. not a pointer) with certain types of arguments and a certain return value.

**args:** list[*RustSimType*]

**returnty:** *RustSimType* | None

**base:** bool = False

**__init__**(*args*, *returnty*, *label=None*, *arg_names=None*, *variadic=False*, *is_arg0_retbuf=False*, *is_class_member_function=False*)

**Parameters**

- **label** – The type label
- **args** (list[*RustSimType*]) – A tuple of types representing the arguments to the function
- **returnty** (*RustSimType* | None) – The return type of the function, or none for void
- **variadic** – Whether the function accepts varargs

**property size**

The size of the type in bits, or None if no size is computable.

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int* | *None*)

**normalize**()

**copy**()

class angr.rust.sim_type.**RustSimTypeReference**

Bases: *RustSimType*, *SimTypePointer*

SimTypePointer is a type that specifies a pointer to some other type.

**__init__**(*pts_to*, *label=None*, *offset=0*)

**Parameters**

- **label** – The type label.
- **pts_to** – The type to which this pointer points.

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int* | *None*)

**property size**

The size of the type in bits, or None if no size is computable.

**copy**()

class angr.rust.sim_type.**RustSimTypeArray**

Bases: *RustSimType*, *SimTypeArray*

SimTypeArray is a type that specifies a series of data laid out in sequence.

__init__(*elem_type*, *length=None*, *label=None*)

**Parameters**

- **label** – The type label.
- **elem_type** – The type of each element in the array.
- **length** – An expression of the length of the array, if known.

repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

copy()

property size

The size of the type in bits, or None if no size is computable.

class angr.rust.sim_type.**RustSimStruct**

Bases: *RustSimType*, *SimStruct*

property size

The size of the type in bits, or None if no size is computable.

repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

copy()

get_field_ty(*name*)

get_field_offset(*name*, *default=None*)

class angr.rust.sim_type.**RustSimTypeNumOffset**

Bases: *RustSimType*, *SimTypeNumOffset*

like SimTypeNum, but supports an offset of 1 to 7 to a byte aligned address to allow structs with bitfields

repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

class angr.rust.sim_type.**RustSimTypeSlice**

Bases: *RustSimStruct*, *SimType*

repr(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

copy()

property size

The size of the type in bits, or None if no size is computable.

class angr.rust.sim_type.**RustSimTypeVec**

Bases: *RustSimStruct*, *SimType*

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

class angr.rust.sim_type.**RustSimTypeBottom**

Bases: *RustSimType*, *SimTypeBottom*

**property size: int**

The size of the type in bits, or None if no size is computable.

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

class angr.rust.sim_type.**EnumVariant**

Bases: object

**__init__**(*name*, *fields*, *discriminant*, *discriminant_size*)

**fields: list[tuple[SimType, str | None]]**

**static from_no_data**(*name*, *discriminant*, *discriminant_size*)

**static from_single_field_ty**(*name*, *field_ty*, *discriminant*, *discriminant_size*)

**has_fields**()

**property first_field_offset**

**property field_offsets**

**property bits**

**property size**

**property type**

**as_struct_ty**()

**with_arch**(*arch*, *memo=None*)

**Parameters**
**memo** (*dict[str, SimType] | None*)

**to_json**(*memo=None*)

**static from_json**(*d*)

class angr.rust.sim_type.**RustSimEnum**

Bases: *RustSimType*, *SimType*

**property size: int**

The size of the type in bits, or None if no size is computable.

**copy**()

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

**get_variant**(*discriminant*)

**Return type**
*EnumVariant* | None

**get_variant_by_name**(*name*)

**num_variants**()

**as_struct_ty**()

class angr.rust.sim_type.**RustSimTypeOption**

Bases: *RustSimEnum*

**copy**()

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

class angr.rust.sim_type.**RustSimTypeResult**

Bases: *RustSimEnum*

**copy**()

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

**Parameters**
**indent** (*int | None*)

class angr.rust.sim_type.**RustSimTypeUnit**

Bases: *RustSimStruct*

**copy**()

**property size**

The size of the type in bits, or None if no size is computable.

class angr.rust.sim_type.**RustSimTypeStrRef**

Bases: *RustSimTypeSlice*

**copy**()

### 10.30.6 angr.rust.structuring

#### Submodules

*structurer_nodes*

#### angr.rust.structuring.structurer_nodes

class angr.rust.structuring.structurer_nodes.**PatternMatchNode**

Bases: *BaseNode*

A Rust match expression node in the structured code tree.

**__init__**(*scrutinee*, *arms*, *default_node*, *addr=None*)

**Parameters**

**arms** (*OrderedDict[tuple[EnumVariant, tuple[Statement]], SequenceNode]*)

**scrutinee**

**arms**

**default_node**

**dbg_repr**(*indent=0*)

class angr.rust.structuring.structurer_nodes.**IfLetNode**

Bases: *BaseNode*

A Rust if-let expression node in the structured code tree.

**__init__**(*pattern*, *scrutinee*, *true_node*, *false_node=None*, *addr=None*)

**pattern**

**scrutinee**

**true_node**

**false_node**

**dbg_repr**(*indent=0*)

### 10.30.7 angr.rust.typehoon

#### Submodules

*translator*
*typehoon*

#### angr.rust.typehoon.translator

class angr.rust.typehoon.translator.**RustSimTypeTempRef**

Bases: *RustSimType*, *SimTypeTempRef*

Temporary reference to an unresolved type variable during translation.

**repr**(*name=None*, *full=0*, *memo=None*, *indent=0*)

class angr.rust.typehoon.translator.**RustTypeTranslator**

Bases: *TypeTranslator*

Bidirectional translator between Rust SimTypes and type constants.

- tc2simtype / concretize: TypeConstant -> RustSimType
- simtype2tc / lift: RustSimType -> TypeConstant

**ctype2rust**(*simtype*)

**Parameters**
**simtype** (SimType)

**arch:** Arch

**func_addr:** int | None

**known_structs:** dict[str, *SimStruct*]

**memo:** dict[str, *Struct*]

**named_struct_id_counter**

**struct_name_to_idx**

**structs**

**translated:** dict[*TypeConstant*, *SimType*]

**translated_simtypes:** dict[*SimType*, *TypeConstant*]

#### angr.rust.typehoon.typehoon

**class** angr.rust.typehoon.typehoon.**RustTypehoon**

Bases: *Typehoon*

Rust-aware type inference engine extending the base Typehoon.

**update_variable_types**(*func_addr*, *var_to_typevars*, *stack_offset_tvs=None*)

**Return type**
None

**Parameters**

- **func_addr** (*int | str*)
- **var_to_typevars** (*dict[SimVariable, set[TypeVariable]]*)
- **stack_offset_tvs** (*dict[int, TypeVariable] | None*)

### 10.30.8 angr.rust.utils

#### Submodules

*ail*
*demangler*
*rust_sigs*

#### angr.rust.utils.ail

**class** angr.rust.utils.ail.**CallFinder**

Bases: *AILBlockViewer*

Find the first Call statement or expression in a block.

**__init__**(*include_macro=False*)

angr.rust.utils.ail.**find_call**(*obj*, *include_macro=False*)

**Parameters**
**obj** (*Block | Statement | Expression*)

angr.rust.utils.ail.**has_call**(*obj*, *include_macro=False*)

**Parameters**
**obj** (*Block | Statement | Expression*)

angr.rust.utils.ail.**get_terminal_call**(*block*)

**Parameters**
**block** (*Block*)

angr.rust.utils.ail.**unwrap_stack_vvar_reference**(*expr*)

**Return type**
*VirtualVariable* | None

angr.rust.utils.ail.**unwrap_combo_reg_vvar_reference**(*expr*)

**Return type**
*VirtualVariable* | None

angr.rust.utils.ail.**deref_vvar_and_offset**(*expr*)

If expr is a dereference of a VirtualVariable (possibly with an offset), return the VirtualVariable and offset. Otherwise, return (None, None).

**Return type**
tuple[*VirtualVariable*, int] | tuple[None, None]

angr.rust.utils.ail.**extract_vvar_and_offset**(*expr*)

If expr is a VirtualVariable (possibly with an offset), return the VirtualVariable and offset. Otherwise, return (None, None).

**Return type**
tuple[*VirtualVariable*, int] | tuple[None, None]

angr.rust.utils.ail.**unwrap_stack_vvar_reference_with_offset**(*expr*)

**Return type**
tuple[*VirtualVariable*, int] | tuple[None, None]

**class** angr.rust.utils.ail.**CallVisitor**

Bases: *AILBlockViewer*

Visit all Call statements and expressions in a graph, invoking a callback.

**__init__**(*callback*)

**visit**(*graph*)

### angr.rust.utils.demangler

angr.rust.utils.demangler.**demangle**(*s*)

angr.rust.utils.demangler.**normalize**(*name*, *monopolize=True*, *concise=False*, *use_trait_name=False*)

### angr.rust.utils.rust_sigs

angr.rust.utils.rust_sigs.**get_default_sig_dir**(*arch_name='x86_64'*, *platform='linux'*)

**Return type**
str | None

**Parameters**

- **arch_name** (*str*)
- **platform** (*str*)

## 10.31 angr.rustylib

**class** angr.rustylib.**Segment**

Bases: object

**classmethod __new__**(**args*, ***kwargs*)

**copy**()

**end**

**size**

**sort**

**start**

**class** angr.rustylib.**SegmentList**

Bases: object

Should be called a SegmentMap!

**classmethod __new__**(**args*, ***kwargs*)

**copy**()

**has_blocks**

**is_occupied**(*address*)

**next_free_pos**(*address*)

**next_pos_with_sort_not_in**(*address*, *sorts*, *max_distance=None*)

Returns the next occupied position that is not in the given set of sorts.

**occupied_by**(*address*)

**occupied_by_sort**(*address*)

**occupied_size**

**occupy**(*address*, *size*, *sort*)

**release**(*address*, *size*)

**search**(*addr*)

Checks which segment that the address *addr* should belong to, and returns the offset of that segment. Note that the address may not actually belong to the block.

This is O(n): an index into the map cannot be computed any faster. Prefer the point-query methods (*occupied_by*, *occupied_by_sort*, ...) wherever an index is not strictly needed.

**sort_ratio_backwards**(*addr*, *window_size*, *sort*)

The fraction of bytes belonging to segments of sort *sort*, among the last *window_size* occupied bytes at or before *addr*.

The walk starts at the segment *search()* would return for *addr* and moves backwards, skipping over gaps, until *window_size* bytes have been covered. Returns 0.0 when the starting segment has a different sort, or when fewer than *window_size* occupied bytes are available.

**update**(*other*)

## 10.32 angr.serializable

**class** angr.serializable.**Serializable**

Bases: object

The base class of all protobuf-serializable classes in angr.

**serialize_to_cmessage**()

Serialize the class object and returns a protobuf cmessage object.

**Returns**
A protobuf cmessage object.

**Return type**
protobuf.cmessage

**serialize**()

Serialize the class object and returns a bytes object.

**Returns**
A bytes object.

**Return type**
*bytes*

**classmethod parse_from_cmessage**(*cmsg*, ***kwargs*)

Parse a protobuf cmessage and create a class object.

**Parameters**
**cmsg** – The probobuf cmessage object.

**Returns**
A unserialized class object.

**Return type**
cls

**classmethod parse**(*s*, ***kwargs*)

Parse a bytes object and create a class object.

**Parameters**
**s** (bytes) – A bytes object.

**Returns**
A class object.

**Return type**
cls

## 10.33 angr.sim_manager

**class** angr.sim_manager.**SimulationManager**

Bases: object

The Simulation Manager is the future future.

Simulation managers allow you to wrangle multiple states in a slick way. States are organized into "stashes", which you can step forward, filter, merge, and move around as you wish. This allows you to, for example, step two different stashes of states at different rates, then merge them together.

Stashes can be accessed as attributes (i.e. .active). A mulpyplexed stash can be retrieved by prepending the name with *mp_*, e.g. *.mp_active*. A single state from the stash can be retrieved by prepending the name with *one_*, e.g. *.one_active*.

Note that you shouldn't usually be constructing SimulationManagers directly - there is a convenient shortcut for creating them in `Project.factory`: see *angr.factory.AngrObjectFactory*.

The most important methods you should look at are `step`, `explore`, and `use_technique`.

**Parameters**

- **project** (angr.project.Project) – A Project instance.
- **stashes** – A dictionary to use as the stash store.
- **active_states** – Active states to seed the "active" stash with.
- **hierarchy** – A StateHierarchy object to use to track the relationships between states.
- **resilience** – A set of errors to catch during stepping to put a state in the `errore` list. You may also provide the values False, None (default), or True to catch, respectively, no errors, all angr-specific errors, and a set of many common errors.
- **save_unsat** – Set to True in order to introduce unsatisfiable states into the `unsat` stash instead of discarding them immediately.
- **auto_drop** – A set of stash names which should be treated as garbage chutes.
- **completion_mode** – A function describing how multiple exploration techniques with the `complete` hook set will interact. By default, the builtin function `any`.
- **techniques** – A list of techniques that should be pre-set to use with this manager.
- **suggestions** – Whether to automatically install the Suggestions exploration technique. Default True.

**Variables**

- **errored** – Not a stash, but a list of ErrorRecords. Whenever a step raises an exception that we catch, the state and some information about the error are placed in this list. You can adjust the list of caught exceptions with the *resilience* parameter.
- **stashes** – All the stashes on this instance, as a dictionary.
- **completion_mode** – A function describing how multiple exploration techniques with the `complete` hook set will interact. By default, the builtin function `any`.

ALL = '_ALL'

DROP = '_DROP'

__init__(*project*, *active_states=None*, *stashes=None*, *hierarchy=None*, *resilience=None*, *save_unsat=False*, *auto_drop=None*, *errored=None*, *completion_mode=<built-in function any>*, *techniques=None*, *suggestions=True*, ***kwargs*)

active: list[*SimState*]

stashed: list[*SimState*]

pruned: list[*SimState*]

unsat: list[*SimState*]

deadended: list[*SimState*]

unconstrained: list[*SimState*]

found: list[*SimState*]

one_active: *SimState*

one_stashed: *SimState*

one_pruned: *SimState*

one_unsat: *SimState*

one_deadended: *SimState*

one_unconstrained: *SimState*

one_found: *SimState*

property errored: list[*ErrorRecord*]

property stashes: defaultdict[str, list[*SimState*]]

mulpyplex(**stashes*)

Mulpyplex across several stashes.

**Parameters**
**stashes** – the stashes to mulpyplex

**Returns**
a mulpyplexed list of states from the stashes in question, in the specified order

copy(*deep=False*)

Make a copy of this simulation manager. Pass `deep=True` to copy all the states in it as well.

If the current callstack includes hooked methods, the already-called methods will not be included in the copy.

use_technique(*tech*)

Use an exploration technique with this SimulationManager.

Techniques can be found in *angr.exploration_techniques*.

**Parameters**
**tech** (ExplorationTechnique) – An ExplorationTechnique object that contains code to modify this SimulationManager's behavior.

**Returns**
The technique that was added, for convenience

**remove_technique**(*tech*)

Remove an exploration technique from a list of active techniques.

**Parameters**
**tech** (ExplorationTechnique) – An ExplorationTechnique object.

**explore**(*stash='active'*, *n=None*, *find=None*, *avoid=None*, *find_stash='found'*, *avoid_stash='avoid'*, *cfg=None*, *num_find=1*, *avoid_priority=False*, ***kwargs*)

Tick stash "stash" forward (up to "n" times or until "num_find" states are found), looking for condition "find", avoiding condition "avoid". Stores found states into "find_stash' and avoided states into "avoid_stash".

The "find" and "avoid" parameters may be any of:

- An address to find
- A set or list of addresses to find
- A function that takes a state and returns whether or not it matches.

If an angr CFG is passed in as the "cfg" parameter and "find" is either a number or a list or a set, then any states which cannot possibly reach a success state without going through a failure state will be preemptively avoided.

**run**(*stash='active'*, *n=None*, *until=None*, ***kwargs*)

Run until the SimulationManager has reached a completed state, according to the current exploration techniques. If no exploration techniques that define a completion state are being used, run until there is nothing left to run.

**Parameters**

- **stash** – Operate on this stash
- **n** – Step at most this many times
- **until** – If provided, should be a function that takes a SimulationManager and returns True or False. Stepping will terminate when it is True.

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**complete**()

Returns whether or not this manager has reached a "completed" state.

**step**(*stash='active'*, *target_stash=None*, *n=None*, *selector_func=None*, *step_func=None*, *error_list=None*, *successor_func=None*, *until=None*, *filter_func=None*, ***run_args*)

Step a stash of states forward and categorize the successors appropriately.

The parameters to this function allow you to control everything about the stepping and categorization process.

**Parameters**

- **stash** – The name of the stash to step (default: 'active')
- **target_stash** – The name of the stash to put the results in (default: same as `stash`)
- **error_list** – The list to put ErrorRecord objects in (default: `self.errored`)
- **selector_func** – If provided, should be a function that takes a state and returns a boolean. If True, the state will be stepped. Otherwise, it will be kept as-is.
- **step_func** – If provided, should be a function that takes a SimulationManager and returns a SimulationManager. Will be called with the SimulationManager at every step. Note that this function should not actually perform any stepping - it is meant to be a maintenance function called after each step.
- **successor_func** – If provided, should be a function that takes a state and return its successors. Otherwise, project.factory.successors will be used.
- **filter_func** – If provided, should be a function that takes a state and return the name of the stash, to which the state should be moved.
- **until** – (DEPRECATED) If provided, should be a function that takes a SimulationManager and returns True or False. Stepping will terminate when it is True.
- **n** – (DEPRECATED) The number of times to step (default: 1 if "until" is not provided)

Additionally, you can pass in any of the following keyword args for project.factory.successors:

**Parameters**

- **jumpkind** – The jumpkind of the previous exit
- **addr** – An address to execute at instead of the state's ip.
- **stmt_whitelist** – A list of stmt indexes to which to confine execution.
- **last_stmt** – A statement index at which to stop execution.
- **thumb** – Whether the block should be lifted in ARM's THUMB mode.
- **backup_state** – A state to read bytes from instead of using project memory.
- **opt_level** – The VEX optimization level to use.
- **insn_bytes** – A string of bytes to use for the block instead of the project.
- **size** – The maximum size of the block, in bytes.
- **num_inst** – The maximum number of instructions.
- **traceflags** – traceflags to be passed to VEX. Default: 0

**Returns**

The simulation manager, for chaining.

**Return type**

*SimulationManager*

**step_state**(*state*, *successor_func=None*, *error_list=None*, ***run_args*)

Don't use this function manually - it is meant to interface with exploration techniques.

**filter**(*state*, *filter_func=None*)

Don't use this function manually - it is meant to interface with exploration techniques.

**selector**(*state*, *selector_func=None*)

Don't use this function manually - it is meant to interface with exploration techniques.

**successors**(*state*, *successor_func=None*, ***run_args*)

Don't use this function manually - it is meant to interface with exploration techniques.

**prune**(*filter_func=None*, *from_stash='active'*, *to_stash='pruned'*)

Prune unsatisfiable states from a stash.

This function will move all unsatisfiable states in the given stash into a different stash.

**Parameters**

- **filter_func** – Only prune states that match this filter.
- **from_stash** – Prune states from this stash. (default: 'active')
- **to_stash** – Put pruned states in this stash. (default: 'pruned')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**populate**(*stash*, *states*)

Populate a stash with a collection of states.

**Parameters**

- **stash** – A stash to populate.
- **states** – A list of states with which to populate the stash.

**absorb**(*simgr*)

Collect all the states from `simgr` and put them in their corresponding stashes in this manager. This will not modify `simgr`.

**move**(*from_stash*, *to_stash*, *filter_func=None*)

Move states from one stash to another.

**Parameters**

- **from_stash** – Take matching states from this stash.
- **to_stash** – Put matching states into this stash.
- **filter_func** – Stash states that match this filter. Should be a function that takes a state and returns True or False. (default: stash all states)

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**stash**(*filter_func=None*, *from_stash='active'*, *to_stash='stashed'*)

Stash some states. This is an alias for move(), with defaults for the stashes.

**Parameters**

- **filter_func** – Stash states that match this filter. Should be a function that takes a state and returns True or False. (default: stash all states)
- **from_stash** – Take matching states from this stash. (default: 'active')
- **to_stash** – Put matching states into this stash. (default: 'stashed')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**unstash**(*filter_func=None*, *to_stash='active'*, *from_stash='stashed'*)

Unstash some states. This is an alias for move(), with defaults for the stashes.

**Parameters**

- **filter_func** – Unstash states that match this filter. Should be a function that takes a state and returns True or False. (default: unstash all states)
- **from_stash** – take matching states from this stash. (default: 'stashed')
- **to_stash** – put matching states into this stash. (default: 'active')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**drop**(*filter_func=None*, *stash='active'*)

Drops states from a stash. This is an alias for move(), with defaults for the stashes.

**Parameters**

- **filter_func** – Drop states that match this filter. Should be a function that takes a state and returns True or False. (default: drop all states)
- **stash** – Drop matching states from this stash. (default: 'active')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**apply**(*state_func=None*, *stash_func=None*, *stash='active'*, *to_stash=None*)

Applies a given function to a given stash.

**Parameters**

- **state_func** – A function to apply to every state. Should take a state and return a state. The returned state will take the place of the old state. If the function *doesn't* return a state, the old state will be used. If the function returns a list of states, they will replace the original states.
- **stash_func** – A function to apply to the whole stash. Should take a list of states and return a list of states. The resulting list will replace the stash. If both state_func and stash_func are provided state_func is applied first, then stash_func is applied on the results.
- **stash** – A stash to work with.
- **to_stash** – If specified, this stash will be used to store the resulting states instead.

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**split**(*stash_splitter=None*, *stash_ranker=None*, *state_ranker=None*, *limit=8*, *from_stash='active'*, *to_stash='stashed'*)

Split a stash of states into two stashes depending on the specified options.

The stash from_stash will be split into two stashes depending on the other options passed in. If to_stash is provided, the second stash will be written there.

stash_splitter overrides stash_ranker, which in turn overrides state_ranker. If no functions are provided, the states are simply split according to the limit.

The sort done with state_ranker is ascending.

**Parameters**

- **stash_splitter** – A function that should take a list of states and return a tuple of two lists (the two resulting stashes).
- **stash_ranker** – A function that should take a list of states and return a sorted list of states. This list will then be split according to "limit".
- **state_ranker** – An alternative to stash_splitter. States will be sorted with outputs of this function, which are to be used as a key. The first "limit" of them will be kept, the rest split off.
- **limit** – For use with state_ranker. The number of states to keep. Default: 8
- **from_stash** – The stash to split (default: 'active')
- **to_stash** – The stash to write to (default: 'stashed')

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**merge**(*merge_func=None*, *merge_key=None*, *stash='active'*, *prune=True*)

Merge the states in a given stash.

**Parameters**

- **stash** – The stash (default: 'active')
- **merge_func** – If provided, instead of using state.merge, call this function with the states as the argument. Should return the merged state.
- **merge_key** – If provided, should be a function that takes a state and returns a key that will compare equal for all states that are allowed to be merged together, as a first approximation. By default: uses PC, callstack, and open file descriptors.
- **prune** – Whether to prune the stash prior to merging it

**Returns**
The simulation manager, for chaining.

**Return type**
*SimulationManager*

**class** angr.sim_manager.**ErrorRecord**

Bases: object

A container class for a state and an error that was thrown during its execution. You can find these in SimulationManager.errored.

**Variables**

- **state** (*SimState*) – The state that encountered an error, at the point in time just before the erroring step began.
- **error** (Exception) – The error that was thrown.
- **traceback** (TracebackType) – The traceback for the error that was thrown.

**__init__**(*state*, *error*, *traceback*)

**state:** *SimState*

**error:** *Exception*

**traceback:** TracebackType

**debug**()

Launch a postmortem debug shell at the site of the error.

**reraise**()

## 10.34 angr.sim_options

## 10.35 angr.sim_procedure

**class** angr.sim_procedure.**SimProcedure**

Bases: object

A SimProcedure is a wonderful object which describes a procedure to run on a state.

You may subclass SimProcedure and override `run()`, replacing it with mutating `self.state` however you like, and then either returning a value or jumping away somehow.

A detailed discussion of programming SimProcedures may be found at https://docs.angr.io/extending-angr/simprocedures

**Parameters**

- **arch** – The architecture to use for this procedure
- **symbolic_return** – Whether the procedure's return value should be stubbed into a single symbolic variable constratined to the real return value
- **returns** – Whether the procedure should return to its caller afterwards
- **is_syscall** – Whether this procedure is a syscall
- **num_args** – The number of arguments this procedure should extract
- **display_name** – The name to use when displaying this procedure
- **library_name** – The name of the library from which the function we're emulating comes
- **cc** (*SimCC* | None) – The SimCC to use for this procedure
- **sim_kwargs** – Additional keyword arguments to be passed to run()
- **is_function** – Whether this procedure emulates a function

The following class variables should be set if necessary when implementing a new SimProcedure:

**Variables**

- **NO_RET** – Set this to true if control flow will never return from this function

- **DYNAMIC_RET** – Set this to true if whether the control flow returns from this function or not depends on the context (e.g., libc's error() call). Must implement dynamic_returns() method.
- **ADDS_EXITS** – Set this to true if you do any control flow other than returning
- **IS_FUNCTION** – Does this procedure simulate a function? True by default
- **ARGS_MISMATCH** – Does this procedure have a different list of arguments than what is provided in the function specification? This may happen when we manually extract arguments in the run() method of a SimProcedure. False by default.
- **local_vars** – If you use `self.call()`, set this to a list of all the local variable names in your class. They will be restored on return.

The following instance variables are available when working with simprocedures from the inside or the outside:

**Variables**

- **project** – The associated angr project
- **arch** – The associated architecture
- **addr** – The linear address at which the procedure is executing
- **cc** – The calling convention in use for engaging with the ABI
- **canonical** – The canonical version of this SimProcedure. Procedures are deepcopied for many reasons, including to be able to store state related to a specific run and to be able to hook continuations.
- **kwargs** – Any extra keyword arguments used to construct the procedure; will be passed to `run`
- **display_name** – See the eponymous parameter
- **library_name** – See the eponymous parameter
- **abi** – If this is a syscall simprocedure, which ABI are we using to map the syscall numbers?
- **symbolic_return** – See the eponymous parameter
- **syscall_number** – If this procedure is a syscall, the number will be populated here.
- **returns** – See eponymous parameter and NO_RET cvar
- **is_syscall** – See eponymous parameter
- **is_function** – See eponymous parameter and cvar
- **is_stub** – See eponymous parameter
- **is_continuation** – Whether this procedure is the original or a continuation resulting from `self.call()`
- **continuations** – A mapping from name to each known continuation
- **run_func** – The name of the function implementing the procedure. "run" by default, but different in continuations.
- **num_args** – The number of arguments to the procedure. If not provided in the parameter, extracted from the definition of `self.run`

The following instance variables are only used in a copy of the procedure that is actually executing on a state:

**Variables**

- **state** – The SimState we should be mutating to perform the procedure

- **successors** – The SimSuccessors associated with the current step
- **arguments** – The function arguments, deserialized from the state
- **arg_session** (*ArgSession* | int | None) – The ArgSession that was used to parse arguments out of the state, in case you need it for varargs
- **use_state_arguments** – Whether we're using arguments extracted from the state or manually provided
- **ret_to** – The current return address
- **ret_expr** – The computed return value
- **call_ret_expr** – The return value from having used `self.call()`
- **inhibit_autoret** – Whether we should avoid automatically adding an exit for returning once the run function ends
- **arg_session** – The ArgSession object that was used to extract the runtime argument values. Useful for if you want to extract variadic args.

**__init__**(*project=None*, *cc=None*, *prototype=None*, *symbolic_return=None*, *returns=None*, *is_syscall=False*, *is_stub=False*, *num_args=None*, *display_name=None*, *library_name=None*, *is_function=None*, ***kwargs*)

**Parameters**

- **project** (Project | *None*)
- **cc** (SimCC | *None*)
- **prototype** (*str* | SimTypeFunction | *None*)

**state:** *SimState*

**arg_session:** *ArgSession* | int | None

**execute**(*state*, *successors=None*, *arguments=None*, *ret_to=None*)

Call this method with a SimState and a SimSuccessors to execute the procedure.

Alternately, successors may be none if this is an inline call. In that case, you should provide arguments to the function.

**make_continuation**(*name*)

**NO_RET = False**

**DYNAMIC_RET = False**

**ADDS_EXITS = False**

**IS_FUNCTION = True**

**ARGS_MISMATCH = False**

**ALT_NAMES = None**

**local_vars:** tuple[str, ...] = ()

**run**(**args*, ***kwargs*)

Implement the actual procedure here!

**Return type**
Any

**static_exits**(*blocks*, ***kwargs*)

Get new exits by performing static analysis and heuristics. This is a fast and best-effort approach to get new exits for scenarios where states are not available (e.g. when building a fast CFG).

**Parameters**
**blocks** (*list*) – Blocks that are executed before reaching this SimProcedure.

**Returns**
A list of dicts. Each dict should contain the following entries: 'address', 'jumpkind', and 'namehint'.

**Return type**
list

**dynamic_returns**(*blocks*, ***kwargs*)

Determines if a call to this function returns or not by performing static analysis and heuristics.

**Parameters**
**blocks** – Blocks that are executed before reaching this SimProcedure.

**Return type**
bool

**Returns**
True if the call returns, False otherwise.

**property should_add_successors**

**set_args**(*args*)

**va_arg**(*ty*, *index=None*)

**inline_call**(*procedure*, **arguments*, ***kwargs*)

Call another SimProcedure in-line to retrieve its return value. Returns an instance of the procedure with the ret_expr property set.

**Parameters**

- **procedure** – The class of the procedure to execute
- **arguments** – Any additional positional args will be used as arguments to the procedure call
- **sim_kwargs** – Any additional keyword args will be passed as sim_kwargs to the procedure constructor

**fix_prototype_returnty**(*ret_size*)

**ret**(*expr=None*)

Add an exit representing a return from this function. If this is not an inline call, grab a return address from the state and jump to it. If this is not an inline call, set a return expression with the calling convention.

**call**(*addr*, *args*, *continue_at*, *cc=None*, *prototype=None*, *jumpkind='Ijk_Call'*)

Add an exit representing calling another function via pointer.

**Parameters**

- **addr** – The address of the function to call
- **args** – The list of arguments to call the function with
- **continue_at** – Later, when the called function returns, execution of the current procedure will continue in the named method.
- **cc** – Optional: use this calling convention for calling the new function. Default is to use the current convention.
- **prototype** – Optional: The prototype to use for the call. Will default to all-ints.

**jump**(*addr*, *jumpkind='Ijk_Boring'*)

Add an exit representing jumping to an address.

**exit**(*exit_code*)

Add an exit representing terminating the program.

**property is_java**

**property argument_types**

**property return_type**

## 10.36 angr.sim_state

angr.sim_state.**arch_overridable**(*f*)

**class** angr.sim_state.**SimState**

Bases: *PluginHub*[*SimStatePlugin*], Generic

The SimState represents the state of a program, including its memory, registers, and so forth.

**Parameters**

**project** (*Project*) – The project instance.

**Variables**

- **regs** – A convenient view of the state's registers, where each register is a property
- **mem** – A convenient view of the state's memory, a *angr.state_plugins.view.SimMemView*
- **registers** – The state's register file as a flat memory region
- **memory** – The state's memory as a flat memory region
- **solver** – The symbolic solver and variable manager for this state
- **inspect** – The breakpoint manager, a *angr.state_plugins.inspect.SimInspector*
- **log** – Information about the state's history
- **scratch** – Information about the current execution step
- **posix** – MISNOMER: information about the operating system or environment model
- **fs** – The current state of the simulated filesystem
- **libc** – Information about the standard library we are emulating
- **cgc** – Information about the cgc environment
- **uc_manager** – Control of under-constrained symbolic execution

- **unicorn** – Control of the Unicorn Engine

**solver:** *SimSolver*

**posix:** *SimSystemPosix*

**fs:** *SimFilesystem*

**libc:** *SimStateLibc*

**registers:** *DefaultMemory*

**regs:** *SimRegNameView*

**memory:** *DefaultMemory*

**callstack:** *CallStack*

**mem:** *SimMemView*

**history:** *SimStateHistory*

**inspect:** *SimInspector*

**jni_references:** *SimStateJNIReferences*

**scratch:** *SimStateScratch*

**heap:** *SimHeapBase*

**__init__**(*project*, *plugins=None*, *mode=None*, *options=None*, *add_options=None*, *remove_options=None*, *special_memory_filler=None*, *os_name=None*, *plugin_preset='default'*, *cle_memory_backer=None*, *dict_memory_backer=None*, *permissions_map=None*, *default_permissions=3*, *stack_perms=None*, *stack_end=None*, *stack_size=None*, *regioned_memory_cls=None*, ***kwargs*)

**Parameters**

- **project** (Project)
- **plugins** (*dict[str, SimStatePlugin] | None*)
- **mode** (*str | None*)
- **options** (*set[str] | list[str] | SimStateOptions | None*)
- **add_options** (*set[str] | None*)
- **remove_options** (*set[str] | None*)
- **special_memory_filler** (*Callable[[str, int, int, SimState], Any] | None*)
- **os_name** (*str | None*)
- **plugin_preset** (*str*)
- **cle_memory_backer** (*Clemory | None*)
- **dict_memory_backer** (*dict[int, bytes] | None*)
- **permissions_map** (*dict[tuple[int, int], int] | None*)
- **default_permissions** (*int*)

- **stack_perms** (*int | None*)
- **stack_end** (*int | None*)
- **stack_size** (*int | None*)

**property plugins**

**property ip**

Get the instruction pointer expression, trigger SimInspect breakpoints, and generate SimActions. Use `_ip` to not trigger breakpoints or generate actions.

**Returns**
an expression

**property addr: IPTypeConc**

Get the concrete address of the instruction pointer, without triggering SimInspect breakpoints or generating SimActions. An integer is returned, or an exception is raised if the instruction pointer is symbolic.

**Returns**
an int

**property arch: Arch**

The architecture of the state. This is the project's architecture, except for JavaVM projects with JNI support, where it is either the Soot or the native architecture, depending on whether the current ip is a Soot address.

**property javavm_memory**

In case of an JavaVM with JNI support, a state can store the memory plugin twice; one for the native and one for the java view of the state.

**Returns**
The JavaVM view of the memory plugin.

**property javavm_registers**

In case of an JavaVM with JNI support, a state can store the registers plugin twice; one for the native and one for the java view of the state.

**Returns**
The JavaVM view of the registers plugin.

**simplify**(**args*)

Simplify this state's constraints.

**add_constraints**(**constraints*)

Add some constraints to the state.

You may pass in any number of symbolic booleans as variadic positional arguments.

**satisfiable**(***kwargs*)

Whether the state's constraints are satisfiable

**downsize**()

Clean up after the solver engine. Calling this when a state no longer needs to be solved on will reduce memory usage.

**step**(***kwargs*)

Perform a step of symbolic execution using this state. Any arguments to *AngrObjectFactory.successors* can be passed to this.

**Returns**
A SimSuccessors object categorizing the results of the step.

**block**(**args*, ***kwargs*)

Represent the basic block at this state's instruction pointer. Any arguments to *AngrObjectFactory.block* can ba passed to this.

**Returns**
A Block object describing the basic block of code at this point.

**copy**()

Returns a copy of the state.

**merge**(**others*, ***kwargs*)

Merges this state with the other states. Returns the merging result, merged state, and the merge flag.

**Parameters**

- **states** – the states to merge
- **merge_conditions** – a tuple of the conditions under which each state holds
- **common_ancestor** – a state that represents the common history between the states being merged. Usually it is only available when EFFICIENT_STATE_MERGING is enabled, otherwise weak-refed states might be dropped from state history instances.
- **plugin_whitelist** – a list of plugin names that will be merged. If this option is given and is not None, any plugin that is not inside this list will not be merged, and will be created as a fresh instance in the new state.
- **common_ancestor_history** – a SimStateHistory instance that represents the common history between the states being merged. This is to allow optimal state merging when EFFICIENT_STATE_MERGING is disabled.

**Returns**
(merged state, merge flag, a bool indicating if any merging occurred)

**reg_concrete**(**args*, ***kwargs*)

Returns the contents of a register but, if that register is symbolic, raises a SimValueError.

**mem_concrete**(**args*, ***kwargs*)

Returns the contents of a memory but, if the contents are symbolic, raises a SimValueError.

**stack_push**(*thing*)

Push 'thing' to the stack, writing the thing to memory and adjusting the stack pointer.

**stack_pop**()

Pops from the stack and returns the popped thing. The length will be the architecture word size.

**stack_read**(*offset*, *length*, *bp=False*)

Reads length bytes, at an offset into the stack.

**Parameters**

- **offset** – The offset from the stack pointer.
- **length** – The number of bytes to read.
- **bp** – If True, offset from the BP instead of the SP. Default: False.

**make_concrete_int**(*expr*)

**dbg_print_stack**(*depth=None*, *sp=None*)

Only used for debugging purposes. Return the current stack info in formatted string. If depth is None, the current stack frame (from sp to bp) will be printed out.

**set_mode**(*mode*)

**property thumb**

## 10.37 angr.sim_state_options

**class** angr.sim_state_options.**StateOption**

Bases: object

Describes a state option.

**__init__**(*name*, *types*, *default='_NO_DEFAULT_VALUE'*, *description=None*)

**name**

**types**

**default**

**description**

**property has_default_value**

**one_type**()

**class** angr.sim_state_options.**SimStateOptions**

Bases: object

A per-state manager of state options. An option can be either a key-valued entry or a Boolean switch (which can be seen as a key-valued entry whose value can only be either True or False).

```
OPTIONS = {'ABSTRACT_MEMORY': <0 ABSTRACT_MEMORY[bool]>, 'ABSTRACT_SOLVER': <0
ABSTRACT_SOLVER[bool]>, 'ACTION_DEPS': <0 ACTION_DEPS[bool]>, 'ADD_AUTO_REFS': <0
ADD_AUTO_REFS[bool]>, 'ALLOW_SEND_FAILURES': <0 ALLOW_SEND_FAILURES[bool]>,
'ALL_FILES_EXIST': <0 ALL_FILES_EXIST[bool]>, 'ANY_FILE_MIGHT_EXIST': <0
ANY_FILE_MIGHT_EXIST[bool]>, 'APPROXIMATE_FIRST': <0 APPROXIMATE_FIRST[bool]>,
'APPROXIMATE_GUARDS': <0 APPROXIMATE_GUARDS[bool]>, 'APPROXIMATE_MEMORY_INDICES': <0
APPROXIMATE_MEMORY_INDICES[bool]>, 'APPROXIMATE_MEMORY_SIZES': <0
APPROXIMATE_MEMORY_SIZES[bool]>, 'APPROXIMATE_SATISFIABILITY': <0
APPROXIMATE_SATISFIABILITY[bool]>, 'AST_DEPS': <0 AST_DEPS[bool]>, 'AUTO_REFS': <0
AUTO_REFS[bool]>, 'AVOID_MULTIVALUED_READS': <0 AVOID_MULTIVALUED_READS[bool]>,
'AVOID_MULTIVALUED_WRITES': <0 AVOID_MULTIVALUED_WRITES[bool]>,
'BEST_EFFORT_MEMORY_STORING': <0 BEST_EFFORT_MEMORY_STORING[bool]>,
'BYPASS_ERRORED_IRCCALL': <0 BYPASS_ERRORED_IRCCALL[bool]>, 'BYPASS_ERRORED_IROP':
<0 BYPASS_ERRORED_IROP[bool]>, 'BYPASS_ERRORED_IRSTMT': <0
BYPASS_ERRORED_IRSTMT[bool]>, 'BYPASS_UNSUPPORTED_IRCCALL': <0
BYPASS_UNSUPPORTED_IRCCALL[bool]>, 'BYPASS_UNSUPPORTED_IRDIRTY': <0
BYPASS_UNSUPPORTED_IRDIRTY[bool]>, 'BYPASS_UNSUPPORTED_IREXPR': <0
BYPASS_UNSUPPORTED_IREXPR[bool]>, 'BYPASS_UNSUPPORTED_IROP': <0
BYPASS_UNSUPPORTED_IROP[bool]>, 'BYPASS_UNSUPPORTED_IRSTMT': <0
BYPASS_UNSUPPORTED_IRSTMT[bool]>, 'BYPASS_UNSUPPORTED_SYSCALL': <0
BYPASS_UNSUPPORTED_SYSCALL[bool]>, 'BYPASS_VERITESTING_EXCEPTIONS': <0
BYPASS_VERITESTING_EXCEPTIONS[bool]>, 'CACHELESS_SOLVER': <0
CACHELESS_SOLVER[bool]>, 'CALLLESS': <0 CALLLESS[bool]>, 'CGC_ENFORCE_FD': <0
CGC_ENFORCE_FD[bool]>, 'CGC_NON_BLOCKING_FDS': <0 CGC_NON_BLOCKING_FDS[bool]>,
'CGC_NO_SYMBOLIC_RECEIVE_LENGTH': <0 CGC_NO_SYMBOLIC_RECEIVE_LENGTH[bool]>,
'COMPOSITE_SOLVER': <0 COMPOSITE_SOLVER[bool]>, 'CONCRETIZE': <0 CONCRETIZE[bool]>,
'CONCRETIZE_SYMBOLIC_FILE_READ_SIZES': <0
CONCRETIZE_SYMBOLIC_FILE_READ_SIZES[bool]>, 'CONCRETIZE_SYMBOLIC_WRITE_SIZES': <0
CONCRETIZE_SYMBOLIC_WRITE_SIZES[bool]>, 'CONSERVATIVE_READ_STRATEGY': <0
CONSERVATIVE_READ_STRATEGY[bool]>, 'CONSERVATIVE_WRITE_STRATEGY': <0
CONSERVATIVE_WRITE_STRATEGY[bool]>, 'CONSTRAINT_TRACKING_IN_SOLVER': <0
CONSTRAINT_TRACKING_IN_SOLVER[bool]>, 'COPY_STATES': <0 COPY_STATES[bool]>,
'CPUID_SYMBOLIC': <0 CPUID_SYMBOLIC[bool]>, 'DOWNSIZE_Z3':  <0 DOWNSIZE_Z3[bool]>,
'DO_CCALLS': <0 DO_CCALLS[bool]>, 'DO_RET_EMULATION': <0 DO_RET_EMULATION[bool]>,
'EFFICIENT_STATE_MERGING': <0 EFFICIENT_STATE_MERGING[bool]>, 'ENABLE_NX': <0
ENABLE_NX[bool]>, 'EXCEPTION_HANDLING': <0 EXCEPTION_HANDLING[bool]>,
'EXTENDED_IROP_SUPPORT': <0 EXTENDED_IROP_SUPPORT[bool]>, 'FAST_MEMORY': <0
FAST_MEMORY[bool]>, 'FAST_REGISTERS': <0 FAST_REGISTERS[bool]>, 'FILES_HAVE_EOF': <0
FILES_HAVE_EOF[bool]>, 'HYBRID_SOLVER': <0 HYBRID_SOLVER[bool]>,
'JAVA_IDENTIFY_GETTER_SETTER': <0 JAVA_IDENTIFY_GETTER_SETTER[bool]>,
'JAVA_TRACK_ATTRIBUTES': <0 JAVA_TRACK_ATTRIBUTES[bool]>, 'KEEP_IP_SYMBOLIC': <0
KEEP_IP_SYMBOLIC[bool]>, 'LAZY_SOLVES': <0 LAZY_SOLVES[bool]>,
'MEMORY_CHUNK_INDIVIDUAL_READS': <0 MEMORY_CHUNK_INDIVIDUAL_READS[bool]>,
'MEMORY_FIND_STRICT_SIZE_LIMIT': <0 MEMORY_FIND_STRICT_SIZE_LIMIT[bool]>,
'MEMORY_SYMBOLIC_BYTES_MAP': <0 MEMORY_SYMBOLIC_BYTES_MAP[bool]>,
'NO_CROSS_INSN_OPT': <0 NO_CROSS_INSN_OPT[bool]>, 'NO_IP_CONCRETIZATION': <0
NO_IP_CONCRETIZATION[bool]>, 'NO_SYMBOLIC_JUMP_RESOLUTION': <0
NO_SYMBOLIC_JUMP_RESOLUTION[bool]>, 'NO_SYMBOLIC_SYSCALL_RESOLUTION': <0
NO_SYMBOLIC_SYSCALL_RESOLUTION[bool]>, 'OPTIMIZE_IR': <0 OPTIMIZE_IR[bool]>,
'PRODUCE_ZERODIV_SUCCESSORS': <0 PRODUCE_ZERODIV_SUCCESSORS[bool]>,
'REGION_MAPPING': <0 REGION_MAPPING[bool]>, 'REPLACEMENT_SOLVER': <0
REPLACEMENT_SOLVER[bool]>, 'REVERSE_MEMORY_HASH_MAP': <0
REVERSE_MEMORY_HASH_MAP[bool]>, 'REVERSE_MEMORY_NAME_MAP': <0
REVERSE_MEMORY_NAME_MAP[bool]>, 'RUN_HOOKS_AT_PLT': <0 RUN_HOOKS_AT_PLT[bool]>,
'SHORT_READS': <0 SHORT_READS[bool]>, 'SIMPLIFY_CONSTRAINTS': <0
SIMPLIFY_CONSTRAINTS[bool]>, 'SIMPLIFY_EXIT_GUARD': <0 SIMPLIFY_EXIT_GUARD[bool]>,
'SIMPLIFY_EXIT_STATE': <0 SIMPLIFY_EXIT_STATE[bool]>, 'SIMPLIFY_EXIT_TARGET': <0
SIMPLIFY_EXIT_TARGET[bool]>, 'SIMPLIFY_EXPRS': <0 SIMPLIFY_EXPRS[bool]>,
'SIMPLIFY_MEMORY_READS': <0 SIMPLIFY_MEMORY_READS[bool]>, 'SIMPLIFY_MEMORY_WRITES':
<0 SIMPLIFY_MEMORY_WRITES[bool]>, 'SIMPLIFY_MERGED_CONSTRAINTS': <0
```

**__init__**(*thing*)

**Parameters**
**thing** – Either a set of Boolean switches to enable, or an existing SimStateOptions instance.

**add**(*boolean_switch*)

[COMPATIBILITY] Enable a Boolean switch.

**Parameters**
**boolean_switch** (*str*) – Name of the Boolean switch.

**Returns**
None

**update**(*boolean_switches*)

[COMPATIBILITY] In order to be compatible with the old interface, you can enable a collection of Boolean switches at the same time by doing the following:

```
>>> state.options.update({sim_options.SYMBOLIC, sim_options.ABSTRACT_MEMORY})
```

or

```
>>> state.options.update(sim_options.unicorn)
```

**Parameters**
**boolean_switches** (*set*) – A collection of Boolean switches to enable.

**Returns**
None

**remove**(*name*)

Drop a state option if it exists, or raise a KeyError if the state option is not set.

[COMPATIBILITY] Remove a Boolean switch.

**Parameters**
**name** (*str*) – Name of the state option.

**Returns**
NNone

**discard**(*name*)

Drop a state option if it exists, or silently return if the state option is not set.

[COMPATIBILITY] Disable a Boolean switch.

**Parameters**
**name** (*str*) – Name of the Boolean switch.

**Returns**
None

**difference**(*boolean_switches*)

[COMPATIBILITY] Make a copy of the current instance, and then discard all options that are in boolean_switches.

**Parameters**
**boolean_switches** (*set*) – A collection of Boolean switches to disable.

**Returns**
A new SimStateOptions instance.

**copy**()

Get a copy of the current SimStateOptions instance.

**Returns**
A new SimStateOptions instance.

**Return type**
*SimStateOptions*

**tally**(*exclude_false=True*, *description=False*)

Return a string representation of all state options.

**Parameters**

- **exclude_false** (*bool*) – Whether to exclude Boolean switches that are disabled.
- **description** (*bool*) – Whether to display the description of each option.

**Returns**
A string representation.

**Return type**
str

**classmethod register_option**(*name*, *types*, *default=None*, *description=None*)

Register a state option.

**Parameters**

- **name** (*str*) – Name of the state option.
- **types** – A collection of allowed types of this state option.
- **default** – The default value of this state option.
- **description** (*str*) – The description of this state option.

**Returns**
None

**classmethod register_bool_option**(*name*, *description=None*)

Register a Boolean switch as state option. This is equivalent to cls.register_option(name, set([bool]), description=description)

**Parameters**

- **name** (*str*) – Name of the state option.
- **description** (*str*) – The description of this state option.

**Returns**
None

## 10.38 angr.sim_type

**class** angr.sim_type.**SimType**

Bases: object

SimType exists to track type information for SimProcedures.

**base: bool = True**

**__init__**(*label=None*, *qualifier=None*)

**Parameters**

- **label** – the type label.
- **qualifier** (*Iterable[str] | None*)

**property size: int | None**

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**with_arch**(*arch*, *memo=None*)

**Return type**
*SimType*

**Parameters**

- **arch** (*Arch | None*)
- **memo** (*dict[str, SimType] | None*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**

- **indent** (*int | None*)
- **name_parens** (*bool*)

**copy**()

**extract**(*state*, *addr*, *concrete=False*)

**Return type**
Any

**Parameters**

- **state** (SimState)
- **concrete** (*bool*)

**store**(*state*, *addr*, *value*)

**Parameters**

- **state** (SimState)
- **value** (*Any*)

**extract_claripy**(*bits*)

Given a bitvector *bits* which was loaded from memory in a big-endian fashion, return a more appropriate or structured representation of the data.

A type must have an arch associated in order to use this method.

**Return type**
Any

**to_json**(*fields=None*, *memo=None*)

Serialize the type class to a JSON-compatible dictionary.

**Return type**
dict[str, Any]

**Parameters**

- **fields** (*Iterable[str] | None*)
- **memo** (*dict[str, SimTypeRef] | None*)

**static from_json**(*d*, *type_collection=None*, *memo=None*, *decoded=None*)

Deserialize a type class from a JSON-compatible dictionary.

**Parameters**

- **type_collection** (*SimTypeCollection* | None) – Resolve type references against this collection.
- **memo** (set[str] | None) – Names of types that are being loaded from the type collection (recursion guard).
- **decoded** (dict[str, *SimType*] | None) – Named structs decoded so far in this document. to_json() emits a reference for every repeated occurrence of a named struct, which is resolved here.
- **d** (*dict[str, Any]*)

**class** angr.sim_type.**TypeRef**

Bases: *SimType*

A TypeRef is a reference to a type with a name. This allows for interactivity in type analysis, by storing a type and having the option to update it later and have all references to it automatically update as well.

**property ty**

**property type**

**property name**

This is a read-only property because it is desirable to store typerefs in a mapping from name to type, and we want the mapping to be in the loop for any updates.

**property size**

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**with_arch**(*arch*, *memo=None*)

**Parameters**
**memo** (*dict[str, SimType] | None*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**copy**()

class angr.sim_type.**NamedTypeMixin**

Bases: object

SimType classes with this mixin in the class hierarchy allows setting custom class names. A typical use case is to represent same or similar type classes with different qualified names, such as "std::basic_string" vs "std::__cxx11::basic_string". In such cases, .name stores the qualified name, and .unqualified_name() returns the unqualified name of the type.

**__init__**(**args*, *name=None*, ***kwargs*)

**Parameters**
**name** (*str | None*)

property **name: str**

**unqualified_name**(*lang='c++'*)

**Return type**
str

**Parameters**
**lang** (*str*)

class angr.sim_type.**SimTypeBottom**

Bases: *SimType*

SimTypeBottom basically represents a type error.

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**copy**()

class angr.sim_type.**SimTypeTop**

Bases: *SimType*

SimTypeTop represents any type (mostly used with a pointer for void*).

**copy**()

class angr.sim_type.**SimTypeReg**

Bases: *SimType*

SimTypeReg is the base type for all types that are register-sized.

**__init__**(*size*, *label=None*, *qualifier=None*)

**Parameters**

- **label** – the type label.
- **size** (int | None) – the size of the type (e.g. 32bit, 8bit, etc.).
- **qualifier** (*Iterable | None*)

**store**(*state*, *addr*, *value*)

**Parameters**
**value** (*int | BV*)

**copy**()

class angr.sim_type.**SimTypeNum**

Bases: *SimType*

SimTypeNum is a numeric type of arbitrary length

**__init__**(*size*, *signed=True*, *label=None*, *qualifier=None*)

**Parameters**

- **size** (int) – The size of the integer, in bits
- **signed** – Whether the integer is signed or not
- **label** – A label for the type
- **qualifier** (*Iterable | None*)

**property size: int**

The size of the type in bits, or None if no size is computable.

**extract**(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state**, **addr**, **concrete** (Literal[True]) → int

**store**(*state*, *addr*, *value*)

**Parameters**

**value** (*int | BV*)

**copy**()

class angr.sim_type.**SimTypeInt**

Bases: *SimTypeReg*

SimTypeInt is a type that specifies a signed or unsigned C integer.

**__init__**(*signed=True*, *label=None*, *qualifier=None*)

**Parameters**

- **signed** – True if signed, False if unsigned
- **label** – The type label
- **qualifier** (*Iterable | None*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**

**name_parens** (*bool*)

**property size**

The size of the type in bits, or None if no size is computable.

**extract**(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state**, **addr**, **concrete** (Literal[True]) → int

copy()

class angr.sim_type.SimTypeShort

Bases: SimTypeInt

class angr.sim_type.SimTypeLong

Bases: SimTypeInt

class angr.sim_type.SimTypeLongLong

Bases: SimTypeInt

class angr.sim_type.SimTypeFixedSizeInt

Bases: SimTypeInt

The base class for all fixed-size (i.e., the size stays the same on all platforms) integer types. Do not instantiate this class directly.

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**

- **indent** (*int | None*)
- **name_parens** (*bool*)

property size: int

The size of the type in bits, or None if no size is computable.

class angr.sim_type.SimTypeInt128

Bases: SimTypeFixedSizeInt

class angr.sim_type.SimTypeInt256

Bases: SimTypeFixedSizeInt

class angr.sim_type.SimTypeInt512

Bases: SimTypeFixedSizeInt

class angr.sim_type.SimTypeChar

Bases: SimTypeReg

SimTypeChar is a type that specifies a character; this could be represented by a byte, but this is meant to be interpreted as a character.

__init__(*signed=True*, *label=None*, *qualifier=None*)

**Parameters**

- **label** – the type label.
- **qualifier** (*Iterable | None*)

store(*state*, *addr*, *value*)

**Parameters**
**value** (*int | BV*)

extract(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state**, **addr**, **concrete** (Literal[True]) → bytes

**Parameters**
**concrete** (*bool*)

**Return type**
*BV* | *bytes*

**copy**()

class angr.sim_type.**SimTypeWideChar**

Bases: *SimTypeReg*

SimTypeWideChar is a type that specifies a wide character (a UTF-16 character).

**__init__**(*signed=True*, *label=None*, *endness=Endness.BE*, *qualifier=None*)

**Parameters**

- **label** – the type label.
- **endness** (*Endness*)
- **qualifier** (*Iterable | None*)

**store**(*state*, *addr*, *value*)

**Parameters**
**value** (*int | BV*)

**extract**(*state*, *addr*, *concrete=False*)

**Return type**
Any

**copy**()

class angr.sim_type.**SimTypeBool**

Bases: *SimTypeReg*

**__init__**(*signed=True*, *label=None*, *qualifier=None*)

**Parameters**

- **label** – the type label.
- **qualifier** (*Iterable | None*)

**store**(*state*, *addr*, *value*)

**Parameters**
**value** (*int | BV | bool*)

**extract**(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.Bool
- **self**, **state**, **addr**, **concrete** (Literal[True]) → bool

**copy**()

class angr.sim_type.**SimTypeFd**

Bases: *SimTypeReg*

SimTypeFd is a type that specifies a file descriptor.

__init__(*label=None*, *qualifier=None*)

**Parameters**

- **label** – the type label
- **qualifier** (*Iterable | None*)

property size

The size of the type in bits, or None if no size is computable.

copy()

extract(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state**, **addr**, **concrete** (Literal[True]) → int

class angr.sim_type.**PointerDisposition**

Bases: IntEnum

IN_OUT = 0

IN = 1

OUT = 2

IN_OUTMAYBE = 3

OUTMAYBE = 4

NONE = 5

UNKNOWN = 6

__new__(*value*)

class angr.sim_type.**SimTypePointer**

Bases: *SimTypeReg*

SimTypePointer is a type that specifies a pointer to some other type.

__init__(*pts_to*, *label=None*, *offset=0*, *qualifier=None*, *disposition=PointerDisposition.UNKNOWN*)

**Parameters**

- **label** – The type label.
- **pts_to** (*SimType*) – The type to which this pointer points.
- **qualifier** (*Iterable[str] | None*)
- **disposition** (PointerDisposition | *int*)

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

make(*pts_to*)

property size
The size of the type in bits, or None if no size is computable.

copy()

extract(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state**, **addr**, **concrete** (Literal[True]) → int

class angr.sim_type.**SimTypeReference**

Bases: *SimTypeReg*

SimTypeReference is a type that specifies a reference to some other type.

**refs:** *SimType*

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

make(*refs*)

property size
The size of the type in bits, or None if no size is computable.

copy()

extract(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state**, **addr**, **concrete** (Literal[True]) → int

class angr.sim_type.**SimTypeArray**

Bases: *SimType*

SimTypeArray is a type that specifies a series of data laid out in sequence.

__init__(*elem_type*, *length=None*, *label=None*, *qualifier=None*)

**Parameters**

- **label** – The type label.
- **elem_type** – The type of each element in the array.
- **length** – An expression of the length of the array, if known.
- **qualifier** (*Iterable | None*)

**elem_type:** *SimType*

**length:** int | None

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**property size**

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**copy**()

**extract**(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → list[Any]
- **self**, **state**, **addr**, **concrete** (Literal[True]) → list[Any]

**store**(*state*, *addr*, *value*)

**Parameters**
**value** (*list[int | BV]*)

angr.sim_type.**SimTypeFixedSizeArray**

alias of *SimTypeArray*

**class** angr.sim_type.**SimTypeString**

Bases: *NamedTypeMixin*, *SimType*

SimTypeString is a type that represents a C-style string, i.e. a NUL-terminated array of bytes.

**__init__**(*length=None*, *label=None*, *name=None*, *qualifier=None*)

**Parameters**

- **label** – The type label.
- **length** (int | None) – An expression of the length of the string, if known.
- **name** (*str | None*)
- **qualifier** (*Iterable | None*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**extract**(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state**, **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state**, **addr**, **concrete** (Literal[True]) → bytes

**Parameters**
**state** (SimState)

**property size**

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**copy**()

class angr.sim_type.**SimTypeWString**

Bases: *NamedTypeMixin*, *SimType*

A wide-character null-terminated string, where each character is 2 bytes.

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**extract**(*state*, *addr*, *concrete=False*)

**store**(*state*, *addr*, *value*)

**property size**

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**copy**()

class angr.sim_type.**SimTypeFunction**

Bases: *SimType*

SimTypeFunction is a type that specifies an actual function (i.e. not a pointer) with certain types of arguments and a certain return value.

**base:** bool = False

**__init__**(*args*, *returnty*, *label=None*, *arg_names=None*, *variadic=False*)

**Parameters**

- **label** – The type label
- **args** (Iterable[*SimType*]) – A tuple of types representing the arguments to the function
- **returnty** (*SimType* | None) – The return type of the function, or none for void
- **variadic** – Whether the function accepts varargs
- **arg_names** (*Iterable[str] | None*)

**args:** tuple[*SimType*, ...]

**returnty:** *SimType* | None

**arg_names:** tuple[str, ...]

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*, *show_void=True*)

**Parameters**

- **name_parens** (*bool*)
- **show_void** (*bool*)

**property size**

The size of the type in bits, or None if no size is computable.

**copy**()

class angr.sim_type.**SimTypeCppFunction**

Bases: *SimTypeFunction*

SimTypeCppFunction is a type that specifies an actual C++-style function with information about arguments, return value, and more C++-specific properties.

**Variables**

- **ctor** – Whether the function is a constructor or not.
- **dtor** – Whether the function is a destructor or not.

**copy**()

class angr.sim_type.**SimTypeLength**

Bases: *SimTypeLong*

SimTypeLength is a type that specifies the length of some buffer in memory.

...I'm not really sure what the original design of this class was going for

**__init__**(*signed=False*, *addr=None*, *length=None*, *label=None*)

**Parameters**

- **signed** – Whether the value is signed or not
- **label** – The type label.
- **addr** – The memory address (expression).
- **length** – The length (expression).

**property size**

The size of the type in bits, or None if no size is computable.

**copy**()

class angr.sim_type.**SimTypeFloat**

Bases: *SimTypeReg*

An IEEE754 single-precision floating point number

**sort = FSORT_32**

**signed = True**

**property size:** *int*

The size of the type in bits, or None if no size is computable.

**extract**(*state*, *addr*, *concrete=False*)

**store**(*state*, *addr*, *value*)

**Parameters**

**value** (*int | BV | FP*)

**copy**()

class angr.sim_type.**SimTypeDouble**

Bases: *SimTypeFloat*

An IEEE754 double-precision floating point number

**sort = FSORT_64**

**property size:** *int*

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**copy**()

**class** angr.sim_type.**SimStruct**

Bases: *NamedTypeMixin*, *SimType*

**fields:** *OrderedDict*[*str*, *SimType*]

**property pack**

**property def_order**

**property align**

**property packed**

**property offsets:** *dict*[*str*, *int*]

**extract**(*state*, *addr*, *concrete=False*)

**Return type**
*SimStructValue*

**fixup_bitfield_offsets**(*arch*)

Fix up the offsets of all SimTypeNumOffset fields to byte aligned addresses. Call this after replacing the fields of an arch-ed struct.

**Return type**
None

**Parameters**
**arch** (*Arch*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**property size**

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**store**(*state*, *addr*, *value*)

**Parameters**
**value** (*int | BV*)

**copy**()

class angr.sim_type.**SimStructValue**

Bases: object

A SimStruct type paired with some real values

**__init__**(*struct*, *values=None*)

**Parameters**

- **struct** – A SimStruct instance describing the type of this struct
- **values** – A mapping from struct fields to values

**property struct**

**copy**()

class angr.sim_type.**SimUnion**

Bases: *NamedTypeMixin*, *SimType*

**fields = ('members', 'name')**

**__init__**(*members*, *name=None*, *label=None*, *qualifier=None*)

**Parameters**

- **members** (dict[str, *SimType*]) – The members of the union, as a mapping name -> type
- **name** – The name of the union
- **qualifier** (*Iterable | None*)

**property size**

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**extract**(*state*, *addr*, *concrete=False*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**

**name_parens** (*bool*)

**copy**()

class angr.sim_type.**SimUnionValue**

Bases: object

A SimStruct type paired with some real values

**__init__**(*union*, *values=None*)

**Parameters**

- **union** – A SimUnion instance describing the type of this union
- **values** – A mapping from union members to values

**copy**()

class angr.sim_type.**SimTypeEnum**

Bases: *NamedTypeMixin*, *SimType*

An enum type with named integer constants.

**Parameters**

- **members** (dict[str, int]) – A mapping from member names to their integer values.
- **base_type** (*SimType* | None) – The underlying integer type (default: SimTypeInt).
- **name** (str | None) – The name of the enum type.

**members:** dict[str, int]

**property base_type:** *SimType*

**property size:** int | None

The size of the type in bits, or None if no size is computable.

**property alignment**

The alignment of the type in bytes.

**resolve**(*value*)

Resolve an integer value to its enum member name.

**Parameters**
**value** (int) – The integer value to resolve.

**Return type**
str | None

**Returns**
The member name if found, None otherwise.

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**copy**()

**extract**(*state*, *addr*, *concrete=False*)

**store**(*state*, *addr*, *value*)

class angr.sim_type.**SimTypeBitfield**

Bases: *NamedTypeMixin*, *SimType*

A bitfield/flags type where values can be combinations of named bit flags.

This is useful for representing flags like PROT_READ | PROT_WRITE where multiple flags can be combined with bitwise OR.

**Parameters**

- **flags** (dict[str, int]) – A mapping from flag names to their bitmask values.
- **base_type** (*SimType* | None) – The underlying integer type (default: SimTypeInt).
- **name** (str | None) – The name of the bitfield type.

**flags:** dict[str, int]

property base_type: *SimType*

property size: int | None

The size of the type in bits, or None if no size is computable.

property alignment

The alignment of the type in bytes.

resolve(*value*)

Resolve an integer value to a combination of flag names.

**Parameters**
**value** (int) – The integer value to resolve.

**Return type**
tuple[list[str], int]

**Returns**
A tuple of (matched_flag_names, unknown_bits). If unknown_bits != 0, there are bits set that don't match any known flag.

render(*value*)

Render an integer value as a combination of flag names.

**Parameters**
**value** (int) – The integer value to render.

**Return type**
str

**Returns**
A string like "PROT_READ | PROT_WRITE" or "PROT_READ | PROT_WRITE | 0x100" if there are unknown bits.

has_unknown_bits(*value*)

Check if a value contains bits that don't match any known flag.

**Parameters**
**value** (int) – The integer value to check.

**Return type**
bool

**Returns**
True if there are unknown bits, False otherwise.

validate(*value*)

Check if a value contains only known flags.

**Parameters**
**value** (int) – The integer value to validate.

**Return type**
bool

**Returns**
True if all bits match known flags, False otherwise.

c_repr(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Parameters**
**name_parens** (*bool*)

**copy**()

**extract**(*state*, *addr*, *concrete=False*)

**store**(*state*, *addr*, *value*)

class angr.sim_type.**SimCppClass**

Bases: *SimStruct*

**property members**

**property size**

The size of the type in bits, or None if no size is computable.

**extract**(*state*, *addr*, *concrete=False*)

**Return type**
*SimCppClassValue*

**store**(*state*, *addr*, *value*)

**Parameters**
**value** (*int | BV*)

**copy**()

class angr.sim_type.**SimCppClassValue**

Bases: *SimStructValue*

A SimCppClass type paired with some real values

**copy**()

class angr.sim_type.**SimTypeNumOffset**

Bases: *SimTypeNum*

like SimTypeNum, but supports an offset of 1 to 7 to a byte aligned address to allow structs with bitfields

**extract**(*state*, *addr*, *concrete=False*)

**Overloads**

- **self**, **state** (SimState), **addr**, **concrete** (Literal[False]) → claripy.ast.BV
- **self**, **state** (SimState), **addr**, **concrete** (Literal[True]) → int

**Parameters**
**state** (SimState)

**store**(*state*, *addr*, *value*)

**copy**()

class angr.sim_type.**SimTypeRef**

Bases: *SimType*

SimTypeRef is a to-be-resolved reference to another SimType.

SimTypeRef is not SimTypeReference.

**property name:** str | None

**set_size**(*v*)

**Parameters**
**v** (*int*)

**c_repr**(*name=None*, *full=0*, *memo=None*, *indent=0*, *name_parens=True*)

**Return type**
str

**Parameters**
**name_parens** (*bool*)

angr.sim_type.**register_types**(*types*)

Pass in some types and they will be registered to the global type store.

The argument may be either a mapping from name to SimType, or a plain SimType. The plain SimType must be either a struct or union type with a name present.

```
>>> register_types(parse_types("typedef int x; typedef float y;"))
>>> register_types(parse_type("struct abcd { int ab; float cd; }"))
```

angr.sim_type.**parse_signature**(*defn*, *predefined_types=None*, *arch=None*)

Parse a single function prototype and return its type

**Return type**
*SimTypeFunction*

angr.sim_type.**parse_defns**(*defn*, *predefined_types=None*, *arch=None*)

Parse a series of C definitions, returns a mapping from variable name to variable type object

**Return type**
dict[str, *SimType*]

angr.sim_type.**parse_types**(*defn*, *predefined_types=None*, *arch=None*)

Parse a series of C definitions, returns a mapping from type name to type object

**Return type**
dict[str, *SimType*]

angr.sim_type.**parse_file**(*defn*, *predefined_types=None*, *arch=None*, *side_effect_types=None*)

Parse a series of C definitions, returns a tuple of two type mappings, one for variable definitions and one for type definitions.

**Return type**
tuple[dict[str, *SimType*], dict[str, *SimType*]]

**Parameters**

- **predefined_types** (*dict[Any, SimType] | None*)
- **side_effect_types** (*dict[Any, SimType] | None*)

angr.sim_type.**type_parser_singleton**()

**Return type**
CParser

angr.sim_type.**parse_type**(*defn*, *predefined_types=None*, *arch=None*)

Parse a simple type expression into a SimType

```
>>> parse_type('int *')
```

angr.sim_type.**parse_type_with_name**(*defn*, *preprocess=True*, *predefined_types=None*, *arch=None*, *side_effect_types=None*)

Parse a simple type expression into a SimType, returning a tuple of the type object and any associated name that might be found in the place a name would go in a type declaration.

```
>>> parse_type_with_name('int *foo')
```

**Parameters**

- **predefined_types** (*dict[Any, SimType] | None*)
- **side_effect_types** (*dict[Any, SimType] | None*)

angr.sim_type.**normalize_cpp_function_name**(*name*)

**Return type**
str

**Parameters**
**name** (*str*)

angr.sim_type.**parse_cpp_file**(*cpp_decl*, *with_param_names=False*)

**Parameters**
**with_param_names** (*bool*)

## 10.39 angr.sim_variable

**class** angr.sim_variable.**SimVariable**

Bases: *Serializable*

The base class for all other classes of variables.

**__init__**(*size*, *ident=None*, *name=None*, *region=None*, *category=None*)

**Parameters**

- **ident** (str | None) – A unique identifier provided by user or the program. Usually a string.
- **name** (str | None) – Name of this variable.
- **size** (*int*)
- **region** (*int | None*)

**ident**

**name**

**region: int | None**

**category: str | None**

**renamed**

**auto_renamed**

**candidate_names**

**size**

**copy**()

**Return type**
Self

**loc_repr**(*arch*)

The representation that shows up in a GUI

**Parameters**
**arch** (*Arch*)

**property is_function_argument**

**property bits: int**

**property key: tuple[str | int | None, ...]**

**clear_hash**()

**class** angr.sim_variable.**SimConstantVariable**

Bases: *SimVariable*

Describes a constant variable.

**value**

**copy**()

**Return type**
*SimConstantVariable*

**property key: tuple[str | int | None, ...]**

**class** angr.sim_variable.**SimTemporaryVariable**

Bases: *SimVariable*

Describes a temporary variable.

**tmp_id**

**copy**()

**Return type**
*SimTemporaryVariable*

**property key: tuple[str | int | None, ...]**

**class** angr.sim_variable.**SimRegisterVariable**

Bases: *SimVariable*

Describes a register variable.

**reg**

**property key: tuple[str | int | None, ...]**

**copy**()

**Return type**
*SimRegisterVariable*

**class** angr.sim_variable.**SimComboRegisterVariable**

Bases: *SimVariable*

A variable that spans multiple registers combined into a single value.

**reg_offsets**

**property bits**

**copy**()

**Return type**
*SimComboRegisterVariable*

**class** angr.sim_variable.**SimMemoryVariable**

Bases: *SimVariable*

Describes a memory variable; the base class for other types of memory variables.

**addr**

**copy**()

**Return type**
*SimMemoryVariable*

**property key:** `tuple[str | int | None, ...]`

**class** angr.sim_variable.**SimStackVariable**

Bases: *SimMemoryVariable*

Describes a stack variable.

**base**

**offset**

**base_addr**

**copy**()

**Return type**
*SimStackVariable*

**property key:** `tuple[str | int | None, ...]`

## 10.40 angr.simos

Manage OS-level configuration.

**class** angr.simos.**SimCGC**

Bases: *SimUserland*

Environment configuration for the CGC DECREE platform

**__init__**(*project*, ***kwargs*)

**state_blank**(*flag_page=None*, *allocate_stack_page_count=256*, ***kwargs*)

**Parameters**

- **flag_page** – Flag page content, either a string or a list of BV8s
- **allocate_stack_page_count** – Number of pages to pre-allocate for stack

**state_entry**(*add_options=None*, ***kwargs*)

**class** angr.simos.**SimJavaVM**

Bases: *SimOS*

**__init__**(**args*, ***kwargs*)

**state_entry**(*args=None*, ***kwargs*)

Create an entry state.

**Parameters**

**args** – List of SootArgument values (optional).

**static generate_symbolic_cmd_line_arg**(*state*)

Generates a new symbolic cmd line argument string. :return: The string reference.

**state_call**(*addr*, **args*, ***kwargs*)

Create a native or a Java call state.

**Parameters**

- **addr** – Soot or native addr of the invoke target.
- **args** – List of SootArgument values.

**static get_default_value_by_type**(*type_*, *state*)

Java specify defaults values for primitive and reference types. This method returns the default value for a given type.

**Parameters**

- **type_** (*str*) – Name of type.
- **state** (SimState) – Current SimState.

**Returns**

Default value for this type.

**static cast_primitive**(*state*, *value*, *to_type*)

Cast the value of primitive types.

**Parameters**

- **value** – Bitvector storing the primitive value.
- **to_type** – Name of the targeted type.

**Returns**

Resized value.

**static init_static_field**(*state*, *field_class_name*, *field_name*, *field_type*)

Initialize the static field with an allocated, but not initialized, object of the given type.

**Parameters**

- **state** – State associated to the field.

- **field_class_name** – Class containing the field.
- **field_name** – Name of the field.
- **field_type** – Type of the field and the new object.

**static get_cmd_line_args**(*state*)

**get_addr_of_native_method**(*soot_method*)

Get address of the implementation from a native declared Java function.

**Parameters**
**soot_method** – Method descriptor of a native declared function.

**Returns**
CLE address of the given method.

**get_native_type**(*java_type*)

Maps the Java type to a SimTypeReg representation of its native counterpart. This type can be used to indicate the (well-defined) size of native JNI types.

**Returns**
A SymTypeReg with the JNI size of the given type.

**property native_arch**

Arch of the native simos.

**Type**
return

**get_native_cc**()

**Returns**
SimCC object for the native simos.

**class** angr.simos.**SimLinux**

Bases: *SimUserland*

OS-specific configuration for *nix-y OSes.

**__init__**(*project*, ***kwargs*)

**state_entry**(*args=None*, *env=None*, *argc=None*, ***kwargs*)

**set_entry_register_values**(*state*)

**state_full_init**(***kwargs*)

**prepare_function_symbol**(*symbol_name*, *basic_addr=None*)

Prepare the address space with the data necessary to perform relocations pointing to the given symbol.

Returns a 2-tuple. The first item is the address of the function code, the second is the address of the relocation target.

**get_segment_register_name**()

**class** angr.simos.**SimOS**

Bases: object

A class describing OS/arch-level configuration.

__init__(*project*, *name=None*)

**Parameters**

- **project** (Project)
- **name** (*str | None*)

**name:** str | None

configure_project()

Configure the project to set up global settings (like SimProcedures).

state_blank(*addr=None*, *initial_prefix=None*, *brk=None*, *stack_end=None*, *stack_size=8388608*, *stdin=None*, *thread_idx=None*, *permissions_backer=None*, ***kwargs*)

Initialize a blank state.

All parameters are optional.

**Parameters**

- **addr** – The execution start address.
- **initial_prefix**
- **stack_end** – The end of the stack (i.e., the byte after the last valid stack address).
- **stack_size** – The number of bytes to allocate for stack space
- **brk** – The address of the process' break.

**Returns**

The initialized SimState.

Any additional arguments will be passed to the SimState constructor

state_entry(***kwargs*)

state_full_init(***kwargs*)

state_call(*addr*, **args*, ***kwargs*)

prepare_call_state(*calling_state*, *initial_state=None*, *preserve_registers=()*, *preserve_memory=()*)

This function prepares a state that is executing a call instruction. If given an initial_state, it copies over all of the critical registers to it from the calling_state. Otherwise, it prepares the calling_state for action.

This is mostly used to create minimalistic for CFG generation. Some ABIs, such as MIPS PIE and x86 PIE, require certain information to be maintained in certain registers. For example, for PIE MIPS, this function transfer t9, gp, and ra to the new state.

prepare_function_symbol(*symbol_name*, *basic_addr=None*)

Prepare the address space with the data necessary to perform relocations pointing to the given symbol

Returns a 2-tuple. The first item is the address of the function code, the second is the address of the relocation target.

handle_exception(*successors*, *engine*, *exception*)

Perform exception handling. This method will be called when, during execution, a SimException is thrown. Currently, this can only indicate a segfault, but in the future it could indicate any unexpected exceptional behavior that can't be handled by ordinary control flow.

The method may mutate the provided SimSuccessors object in any way it likes, or re-raise the exception.

**Parameters**

- **successors** – The SimSuccessors object currently being executed on
- **engine** – The engine that was processing this step
- **exception** – The actual exception object

**syscall**(*state*, *allow_unsupported=True*)

**Return type**
*SimProcedure* | None

**Parameters**

- **state** (SimState)
- **allow_unsupported** (*bool*)

**syscall_abi**(*state*)

**Return type**
str | None

**Parameters**
**state** (SimState)

**syscall_cc**(*state*)

**Return type**
*SimCCSyscall* | None

**Parameters**
**state** (SimState)

**is_syscall_addr**(*addr*)

**Return type**
bool

**syscall_from_addr**(*addr*, *allow_unsupported=True*)

**Return type**
*SimProcedure* | None

**syscall_from_number**(*number*, *allow_unsupported=True*, *abi=None*)

**Return type**
*SimProcedure* | None

**setup_gdt**(*state*, *gdt*)
Write the GlobalDescriptorTable object in the current state memory

**Parameters**

- **state** – state in which to write the GDT
- **gdt** – GlobalDescriptorTable object

**Returns**

**generate_gdt**(*fs*, *gs*, *fs_size=4294967295*, *gs_size=4294967295*)
Generate a GlobalDescriptorTable object and populate it using the value of the gs and fs register

**Parameters**

- **fs** – value of the fs segment register

- **gs** – value of the gs segment register
- **fs_size** – size of the fs segment register
- **gs_size** – size of the gs segment register

**Returns**
gdt a GlobalDescriptorTable object

**class** angr.simos.**SimSnimmucNxp**

Bases: *SimOS*

This class implements the "OS" for a bare-metal firmware used at an imaginary company.

**__init__**(*project*, *name=None*, ***kwargs*)

**Parameters**
**project** (Project)

**class** angr.simos.**SimUefi**

Bases: *SimOS*

Environment for UEFI images. UEFI is a boot-time environment with no process model and no syscall interface, so what distinguishes it from a bare SimOS is its calling conventions: the x86 bindings of the UEFI specification adopt the Microsoft conventions rather than the System V ones, which is what DEFAULT_CC records for "UEFI".

**__init__**(*project*)

**class** angr.simos.**SimUserland**

Bases: *SimOS*

This is a base class for any SimOS that wants to support syscalls.

It uses the CLE kernel object to provide addresses for syscalls. Syscalls will be emulated as a jump to one of these addresses, where a SimProcedure from the syscall library provided at construction time will be executed.

**__init__**(*project*, *syscall_library=None*, *syscall_addr_alignment=4*, ***kwargs*)

**syscall_cc**(*state*)

**Return type**
*SimCCSyscall*

**syscall**(*state*, *allow_unsupported=True*)

Given a state, return the procedure corresponding to the current syscall. This procedure will have .syscall_number, .display_name, and .addr set.

**Parameters**

- **state** – The state to get the syscall number from
- **allow_unsupported** – Whether to return a "dummy" sycall instead of raising an unsupported exception

**syscall_abi**(*state*)

Optionally, override this function to determine which abi is being used for the state's current syscall.

**is_syscall_addr**(*addr*)

Return whether or not the given address corresponds to a syscall implementation.

**syscall_from_addr**(*addr*, *allow_unsupported=True*)

Get a syscall SimProcedure from an address.

**Parameters**

- **addr** – The address to convert to a syscall SimProcedure
- **allow_unsupported** – Whether to return a dummy procedure for an unsupported syscall instead of raising an exception.

**Returns**
The SimProcedure for the syscall, or None if the address is not a syscall address.

**syscall_from_number**(*number*, *allow_unsupported=True*, *abi=None*)

Get a syscall SimProcedure from its number.

**Parameters**

- **number** – The syscall number
- **allow_unsupported** – Whether to return a "stub" syscall for unsupported numbers instead of throwing an error
- **abi** – The name of the abi to use. If None, will assume that the abis have disjoint numbering schemes and pick the right one.

**Returns**
The SimProcedure for the syscall

**class** angr.simos.**SimWindows**

Bases: *SimOS*

Environment for the Windows Win32 subsystem. Does not support syscalls currently.

**__init__**(*project*)

**property is_dump:** bool

**state_entry**(*args=None*, *env=None*, *argc=None*, ***kwargs*)

**syscall**(*state*, *allow_unsupported=True*)

Given a state, return the procedure corresponding to the current syscall. This procedure will have .syscall_number, .display_name, and .addr set.

**Parameters**

- **state** – The state to get the syscall number from
- **allow_unsupported** – Whether to return a "dummy" sycall instead of raising an unsupported exception

**is_syscall_addr**(*addr*)

Return whether or not the given address corresponds to a syscall implementation.

**syscall_from_addr**(*addr*, *allow_unsupported=True*)

Get a syscall SimProcedure from an address.

**Parameters**

- **addr** – The address to convert to a syscall SimProcedure
- **allow_unsupported** – Whether to return a dummy procedure for an unsupported syscall instead of raising an exception.

**Returns**
The SimProcedure for the syscall, or None if the address is not a syscall address.

`get_segment_register_name`()

### Submodules

| |
|---|
| *cgc* |
| *javavm* |
| *linux* |
| *simos* |
| *snimmuc_nxp* |
| *uefi* |
| *userland* |
| *windows* |
| *xbox* |

## 10.40.1 angr.simos.cgc

**class** angr.simos.cgc.**SimCGC**

Bases: *SimUserland*

Environment configuration for the CGC DECREE platform

`__init__`(*project*, ***kwargs*)

`state_blank`(*flag_page=None*, *allocate_stack_page_count=256*, ***kwargs*)

**Parameters**

- **flag_page** – Flag page content, either a string or a list of BV8s
- **allocate_stack_page_count** – Number of pages to pre-allocate for stack

`state_entry`(*add_options=None*, ***kwargs*)

## 10.40.2 angr.simos.javavm

**class** angr.simos.javavm.**SimJavaVM**

Bases: *SimOS*

`__init__`(**args*, ***kwargs*)

`state_entry`(*args=None*, ***kwargs*)

Create an entry state.

**Parameters**
**args** – List of SootArgument values (optional).

**static** `generate_symbolic_cmd_line_arg`(*state*)

Generates a new symbolic cmd line argument string. :return: The string reference.

`state_call`(*addr*, **args*, ***kwargs*)

Create a native or a Java call state.

**Parameters**

- **addr** – Soot or native addr of the invoke target.

- **args** – List of SootArgument values.

**static get_default_value_by_type**(*type_*, *state*)

Java specify defaults values for primitive and reference types. This method returns the default value for a given type.

**Parameters**

- **type_** (*str*) – Name of type.
- **state** (SimState) – Current SimState.

**Returns**
Default value for this type.

**static cast_primitive**(*state*, *value*, *to_type*)

Cast the value of primitive types.

**Parameters**

- **value** – Bitvector storing the primitive value.
- **to_type** – Name of the targeted type.

**Returns**
Resized value.

**static init_static_field**(*state*, *field_class_name*, *field_name*, *field_type*)

Initialize the static field with an allocated, but not initialized, object of the given type.

**Parameters**

- **state** – State associated to the field.
- **field_class_name** – Class containing the field.
- **field_name** – Name of the field.
- **field_type** – Type of the field and the new object.

**static get_cmd_line_args**(*state*)

**get_addr_of_native_method**(*soot_method*)

Get address of the implementation from a native declared Java function.

**Parameters**
**soot_method** – Method descriptor of a native declared function.

**Returns**
CLE address of the given method.

**get_native_type**(*java_type*)

Maps the Java type to a SimTypeReg representation of its native counterpart. This type can be used to indicate the (well-defined) size of native JNI types.

**Returns**
A SymTypeReg with the JNI size of the given type.

**property native_arch**

Arch of the native simos.

**Type**
return

**get_native_cc**()

**Returns**
SimCC object for the native simos.

angr.simos.javavm.**prepare_native_return_state**(*native_state*)

Hook target for native function call returns.

Recovers and stores the return value from native memory and toggles the state, s.t. execution continues in the Soot engine.

Note: Redirection needed for pickling.

### 10.40.3 angr.simos.linux

**class** angr.simos.linux.**SimLinux**

Bases: *SimUserland*

OS-specific configuration for *nix-y OSes.

**__init__**(*project*, ***kwargs*)

**state_entry**(*args=None*, *env=None*, *argc=None*, ***kwargs*)

**set_entry_register_values**(*state*)

**state_full_init**(***kwargs*)

**prepare_function_symbol**(*symbol_name*, *basic_addr=None*)

Prepare the address space with the data necessary to perform relocations pointing to the given symbol.

Returns a 2-tuple. The first item is the address of the function code, the second is the address of the relocation target.

**get_segment_register_name**()

### 10.40.4 angr.simos.simos

**class** angr.simos.simos.**SimOS**

Bases: object

A class describing OS/arch-level configuration.

**__init__**(*project*, *name=None*)

**Parameters**

- **project** (Project)
- **name** (*str | None*)

**name:** *str | None*

**configure_project**()

Configure the project to set up global settings (like SimProcedures).

**state_blank**(*addr=None*, *initial_prefix=None*, *brk=None*, *stack_end=None*, *stack_size=8388608*, *stdin=None*, *thread_idx=None*, *permissions_backer=None*, ***kwargs*)

Initialize a blank state.

All parameters are optional.

**Parameters**

- **addr** – The execution start address.
- **initial_prefix**
- **stack_end** – The end of the stack (i.e., the byte after the last valid stack address).
- **stack_size** – The number of bytes to allocate for stack space
- **brk** – The address of the process' break.

**Returns**
The initialized SimState.

Any additional arguments will be passed to the SimState constructor

**state_entry**(***kwargs*)

**state_full_init**(***kwargs*)

**state_call**(*addr*, **args*, ***kwargs*)

**prepare_call_state**(*calling_state*, *initial_state=None*, *preserve_registers=()*, *preserve_memory=()*)

This function prepares a state that is executing a call instruction. If given an initial_state, it copies over all of the critical registers to it from the calling_state. Otherwise, it prepares the calling_state for action.

This is mostly used to create minimalistic for CFG generation. Some ABIs, such as MIPS PIE and x86 PIE, require certain information to be maintained in certain registers. For example, for PIE MIPS, this function transfer t9, gp, and ra to the new state.

**prepare_function_symbol**(*symbol_name*, *basic_addr=None*)

Prepare the address space with the data necessary to perform relocations pointing to the given symbol

Returns a 2-tuple. The first item is the address of the function code, the second is the address of the relocation target.

**handle_exception**(*successors*, *engine*, *exception*)

Perform exception handling. This method will be called when, during execution, a SimException is thrown. Currently, this can only indicate a segfault, but in the future it could indicate any unexpected exceptional behavior that can't be handled by ordinary control flow.

The method may mutate the provided SimSuccessors object in any way it likes, or re-raise the exception.

**Parameters**

- **successors** – The SimSuccessors object currently being executed on
- **engine** – The engine that was processing this step
- **exception** – The actual exception object

**syscall**(*state*, *allow_unsupported=True*)

**Return type**
*SimProcedure* | None

**Parameters**

- **state** (SimState)
- **allow_unsupported** (*bool*)

**syscall_abi**(*state*)

**Return type**
str | None

**Parameters**
**state** (SimState)

**syscall_cc**(*state*)

**Return type**
*SimCCSyscall* | None

**Parameters**
**state** (SimState)

**is_syscall_addr**(*addr*)

**Return type**
bool

**syscall_from_addr**(*addr*, *allow_unsupported=True*)

**Return type**
*SimProcedure* | None

**syscall_from_number**(*number*, *allow_unsupported=True*, *abi=None*)

**Return type**
*SimProcedure* | None

**setup_gdt**(*state*, *gdt*)

Write the GlobalDescriptorTable object in the current state memory

**Parameters**

- **state** – state in which to write the GDT
- **gdt** – GlobalDescriptorTable object

**Returns**

**generate_gdt**(*fs*, *gs*, *fs_size=4294967295*, *gs_size=4294967295*)

Generate a GlobalDescriptorTable object and populate it using the value of the gs and fs register

**Parameters**

- **fs** – value of the fs segment register
- **gs** – value of the gs segment register
- **fs_size** – size of the fs segment register
- **gs_size** – size of the gs segment register

**Returns**
gdt a GlobalDescriptorTable object

**class** angr.simos.simos.**GlobalDescriptorTable**

Bases: object

GlobalDescriptorTable object to store the GDT table and the segment registers values

**__init__**(*addr*, *limit*, *table*, *gdt_sel*, *cs_sel*, *ds_sel*, *es_sel*, *ss_sel*, *fs_sel*, *gs_sel*)

### 10.40.5 angr.simos.snimmuc_nxp

**class** angr.simos.snimmuc_nxp.**SimSnimmucNxp**

Bases: *SimOS*

This class implements the "OS" for a bare-metal firmware used at an imaginary company.

**__init__**(*project*, *name=None*, ***kwargs*)

**Parameters**
**project** (Project)

### 10.40.6 angr.simos.uefi

**class** angr.simos.uefi.**SimUefi**

Bases: *SimOS*

Environment for UEFI images. UEFI is a boot-time environment with no process model and no syscall interface, so what distinguishes it from a bare SimOS is its calling conventions: the x86 bindings of the UEFI specification adopt the Microsoft conventions rather than the System V ones, which is what DEFAULT_CC records for "UEFI".

**__init__**(*project*)

### 10.40.7 angr.simos.userland

**class** angr.simos.userland.**SimUserland**

Bases: *SimOS*

This is a base class for any SimOS that wants to support syscalls.

It uses the CLE kernel object to provide addresses for syscalls. Syscalls will be emulated as a jump to one of these addresses, where a SimProcedure from the syscall library provided at construction time will be executed.

**__init__**(*project*, *syscall_library=None*, *syscall_addr_alignment=4*, ***kwargs*)

**syscall_cc**(*state*)

**Return type**
*SimCCSyscall*

**syscall**(*state*, *allow_unsupported=True*)

Given a state, return the procedure corresponding to the current syscall. This procedure will have .syscall_number, .display_name, and .addr set.

**Parameters**

- **state** – The state to get the syscall number from
- **allow_unsupported** – Whether to return a "dummy" sycall instead of raising an unsupported exception

**syscall_abi**(*state*)

Optionally, override this function to determine which abi is being used for the state's current syscall.

**is_syscall_addr**(*addr*)

Return whether or not the given address corresponds to a syscall implementation.

**syscall_from_addr**(*addr*, *allow_unsupported=True*)

Get a syscall SimProcedure from an address.

**Parameters**

- **addr** – The address to convert to a syscall SimProcedure
- **allow_unsupported** – Whether to return a dummy procedure for an unsupported syscall instead of raising an exception.

**Returns**
The SimProcedure for the syscall, or None if the address is not a syscall address.

**syscall_from_number**(*number*, *allow_unsupported=True*, *abi=None*)

Get a syscall SimProcedure from its number.

**Parameters**

- **number** – The syscall number
- **allow_unsupported** – Whether to return a "stub" syscall for unsupported numbers instead of throwing an error
- **abi** – The name of the abi to use. If None, will assume that the abis have disjoint numbering schemes and pick the right one.

**Returns**
The SimProcedure for the syscall

### 10.40.8 angr.simos.windows

**class** angr.simos.windows.**SecurityCookieInit**

Bases: Enum

Security cooke initialization value initialization method.

**NONE = 0**

**RANDOM = 1**

**STATIC = 2**

**SYMBOLIC = 3**

**class** angr.simos.windows.**SimWindows**

Bases: *SimOS*

Environment for the Windows Win32 subsystem. Does not support syscalls currently.

**__init__**(*project*)

**property is_dump: bool**

**state_entry**(*args=None*, *env=None*, *argc=None*, ***kwargs*)

**syscall**(*state*, *allow_unsupported=True*)

Given a state, return the procedure corresponding to the current syscall. This procedure will have .syscall_number, .display_name, and .addr set.

**Parameters**

- **state** – The state to get the syscall number from

- **allow_unsupported** – Whether to return a "dummy" sycall instead of raising an unsupported exception

**is_syscall_addr**(*addr*)

Return whether or not the given address corresponds to a syscall implementation.

**syscall_from_addr**(*addr*, *allow_unsupported=True*)

Get a syscall SimProcedure from an address.

**Parameters**

- **addr** – The address to convert to a syscall SimProcedure
- **allow_unsupported** – Whether to return a dummy procedure for an unsupported syscall instead of raising an exception.

**Returns**

The SimProcedure for the syscall, or None if the address is not a syscall address.

**get_segment_register_name**()

### 10.40.9 angr.simos.xbox

**class** angr.simos.xbox.**SimXbox**

Bases: *SimOS*

Environment for the original Xbox subsystem (x86, 32-bit).

**__init__**(*project*)

## 10.41 angr.slicer

**class** angr.slicer.**SimLightState**

Bases: object

Represents a program state. Only used in SimSlicer.

**__init__**(*temps=None*, *regs=None*, *stack_offsets=None*, *options=None*)

**temps**

**regs**

**stack_offsets**

**options**

**class** angr.slicer.**SimSlicer**

Bases: object

A super lightweight intra-IRSB slicing class.

**__init__**(*arch*, *statements*, *target_tmps=None*, *target_regs=None*, *target_stack_offsets=None*, *inslice_callback=None*, *inslice_callback_infodict=None*, *include_imarks=True*)

**Parameters**

**include_imarks** (*bool*)

## 10.42 angr.state_hierarchy

**class** angr.state_hierarchy.**StateHierarchy**

Bases: object

The state hierarchy holds weak references to SimStateHistory objects in a directed acyclic graph. It is useful for queries about a state's ancestry, notably "what is the best ancestor state for a merge among these states" and "what is the most recent unsatisfiable state while using LAZY_SOLVES"

**__init__**()

**get_ref**(*obj*)

**dead_ref**(*ref*)

**defer_cleanup**()

**add_state**(*s*)

**add_history**(*h*)

**simplify**()

**full_simplify**()

**lineage**(*h*)

Returns the lineage of histories leading up to *h*.

**all_successors**(*h*)

**history_successors**(*h*)

**history_predecessors**(*h*)

**history_contains**(*h*)

**unreachable_state**(*state*)

**unreachable_history**(*h*)

**most_mergeable**(*states*)

Find the "most mergeable" set of states from those provided.

**Parameters**
**states** – a list of states

**Returns**
a tuple of: (list of states to merge, those states' common history, list of states to not merge yet)

## 10.43 angr.state_plugins

angr.state_plugins.**BP_IPDB**(*state*)

**Return type**
None

**Parameters**
**state** (SimState)

angr.state_plugins.**BP_IPYTHON**(*state*)

**Return type**
None

**Parameters**
**state** (SimState)

**class** angr.state_plugins.**GDB**

Bases: *SimStatePlugin*

Initialize or update a state from gdb dumps of the stack, heap, registers and data (or arbitrary) segments.

**__init__**(*omit_fp=False*, *adjust_stack=False*)

**Parameters**

- **omit_fp** – The frame pointer register is used for something else. (i.e. –omit_frame_pointer)
- **adjust_stack** – Use different stack addresses than the gdb session (not recommended).

**set_stack**(*stack_dump*, *stack_top*)

Stack dump is a dump of the stack from gdb, i.e. the result of the following gdb command :

`dump binary memory [stack_dump] [begin_addr] [end_addr]`

We set the stack to the same addresses as the gdb session to avoid pointers corruption.

**Parameters**

- **stack_dump** – The dump file.
- **stack_top** – The address of the top of the stack in the gdb session.

**set_heap**(*heap_dump*, *heap_base*)

Heap dump is a dump of the heap from gdb, i.e. the result of the following gdb command:

`dump binary memory [stack_dump] [begin] [end]`

**Parameters**

- **heap_dump** – The dump file.
- **heap_base** – The start address of the heap in the gdb session.

**set_data**(*addr*, *data_dump*)

Update any data range (most likely use is the data segments of loaded objects)

**set_regs**(*regs_dump*)

Initialize register values within the state

**Parameters**
**regs_dump** – The output of `info registers` in gdb.

**class** angr.state_plugins.**CallStack**

Bases: *SimStatePlugin*

Stores the address of the function you're in and the value of SP at the VERY BOTTOM of the stack, i.e. points to the return address.

**__init__**(*call_site_addr=0*, *func_addr=0*, *stack_ptr=0*, *ret_addr=0*, *jumpkind='Ijk_Call'*, *next_frame=None*, *invoke_return_variable=None*)

**Parameters**
**next_frame** (*Self | None*)

**copy_without_tail**(*memo*)

**property current_function_address**

Address of the current function.

**Returns**
the address of the function

**Return type**
int

**property current_stack_pointer**

Get the value of the stack pointer.

**Returns**
Value of the stack pointer

**Return type**
int

**property current_return_target**

Get the return target.

**Returns**
The address of return target.

**Return type**
int

**static stack_suffix_to_string**(*stack_suffix*)

Convert a stack suffix to a human-readable string representation. :type tuple stack_suffix: :param tuple stack_suffix: The stack suffix. :return: A string representation :rtype: str

**property top: Self**

Returns the element at the top of the callstack without removing it.

**Returns**
A CallStack.

**push**(*cf*)

Push the frame cf onto the stack. Return the new stack.

**Return type**
Self

**Parameters**
**cf** (*Self*)

**pop**()

Pop the top frame from the stack. Return the new stack.

**call**(*callsite_addr*, *addr*, *retn_target=None*, *stack_pointer=None*)

Push a stack frame into the call stack. This method is called when calling a function in CFG recovery.

**Parameters**

- **callsite_addr** (*int*) – Address of the call site
- **addr** (*int*) – Address of the call target
- **retn_target** (*int or None*) – Address of the return target
- **stack_pointer** (*int*) – Value of the stack pointer

**Returns**
None

**ret**(*retn_target=None*)

Pop one or many call frames from the stack. This method is called when returning from a function in CFG recovery.

**Parameters**
**retn_target** (*int*) – The target to return to.

**Returns**
None

**dbg_repr**()

Debugging representation of this CallStack object.

**Returns**
Details of this CalLStack

**Return type**
str

**stack_suffix**(*context_sensitivity_level*)

Generate the stack suffix. A stack suffix can be used as the key to a SimRun in CFG recovery.

**Parameters**
**context_sensitivity_level** (*int*) – Level of context sensitivity.

**Returns**
A tuple of stack suffix.

**Return type**
tuple[int | None, ...]

**class** angr.state_plugins.**InspectAttrs**

Bases: object

Per-event attributes published by the inspect machinery while a breakpoint is firing.

Each field is set on the active state before any matching breakpoint is checked, then cleared (back to None) after the event completes. Breakpoint actions read and write these fields through state.inspect.attrs to observe or override the in-flight event.

**vex_lift_addr**: Any = None

**vex_lift_size**: Any = None

**vex_lift_buff**: Any = None

**mem_read_address**: Any = None

**mem_read_expr**: Any = None

**mem_read_length**: Any = None

```
    mem_read_condition: Any = None
    mem_read_endness: Any = None
    mem_write_address: Any = None
    mem_write_expr: Any = None
    mem_write_length: Any = None
    mem_write_condition: Any = None
    mem_write_endness: Any = None
    reg_read_offset: Any = None
    reg_read_expr: Any = None
    reg_read_length: Any = None
    reg_read_condition: Any = None
    reg_read_endness: Any = None
    reg_write_offset: Any = None
    reg_write_expr: Any = None
    reg_write_length: Any = None
    reg_write_condition: Any = None
    reg_write_endness: Any = None
    tmp_read_num: Any = None
    tmp_read_expr: Any = None
    tmp_write_num: Any = None
    tmp_write_expr: Any = None
    expr: Any = None
    expr_result: Any = None
    statement: Any = None
    instruction: Any = None
    address: Any = None
    added_constraints: Any = None
    function_address: Any = None
    exit_target: Any = None
    exit_guard: Any = None
    exit_jumpkind: Any = None
```

```
    backtrace: Any = None
    symbolic_name: Any = None
    symbolic_size: Any = None
    symbolic_expr: Any = None
    address_concretization_strategy: Any = None
    address_concretization_action: Any = None
    address_concretization_memory: Any = None
    address_concretization_expr: Any = None
    address_concretization_result: Any = None
    address_concretization_add_constraints: Any = None
    syscall_name: Any = None
    simprocedure_name: Any = None
    simprocedure_addr: Any = None
    simprocedure_result: Any = None
    simprocedure: Any = None
    dirty_name: Any = None
    dirty_handler: Any = None
    dirty_args: Any = None
    dirty_result: Any = None
    sim_engine: Any = None
    sim_successors: Any = None
    mapped_page: Any = None
    mapped_address: Any = None
```

__init__(*vex_lift_addr=None*, *vex_lift_size=None*, *vex_lift_buff=None*, *mem_read_address=None*, *mem_read_expr=None*, *mem_read_length=None*, *mem_read_condition=None*, *mem_read_endness=None*, *mem_write_address=None*, *mem_write_expr=None*, *mem_write_length=None*, *mem_write_condition=None*, *mem_write_endness=None*, *reg_read_offset=None*, *reg_read_expr=None*, *reg_read_length=None*, *reg_read_condition=None*, *reg_read_endness=None*, *reg_write_offset=None*, *reg_write_expr=None*, *reg_write_length=None*, *reg_write_condition=None*, *reg_write_endness=None*, *tmp_read_num=None*, *tmp_read_expr=None*, *tmp_write_num=None*, *tmp_write_expr=None*, *expr=None*, *expr_result=None*, *statement=None*, *instruction=None*, *address=None*, *added_constraints=None*, *function_address=None*, *exit_target=None*, *exit_guard=None*, *exit_jumpkind=None*, *backtrace=None*, *symbolic_name=None*, *symbolic_size=None*, *symbolic_expr=None*, *address_concretization_strategy=None*, *address_concretization_action=None*, *address_concretization_memory=None*, *address_concretization_expr=None*, *address_concretization_result=None*, *address_concretization_add_constraints=None*, *syscall_name=None*, *simprocedure_name=None*, *simprocedure_addr=None*, *simprocedure_result=None*, *simprocedure=None*, *dirty_name=None*, *dirty_handler=None*, *dirty_args=None*, *dirty_result=None*, *sim_engine=None*, *sim_successors=None*, *mapped_page=None*, *mapped_address=None*)

**Parameters**

- **vex_lift_addr** (*Any*)
- **vex_lift_size** (*Any*)
- **vex_lift_buff** (*Any*)
- **mem_read_address** (*Any*)
- **mem_read_expr** (*Any*)
- **mem_read_length** (*Any*)
- **mem_read_condition** (*Any*)
- **mem_read_endness** (*Any*)
- **mem_write_address** (*Any*)
- **mem_write_expr** (*Any*)
- **mem_write_length** (*Any*)
- **mem_write_condition** (*Any*)
- **mem_write_endness** (*Any*)
- **reg_read_offset** (*Any*)
- **reg_read_expr** (*Any*)
- **reg_read_length** (*Any*)
- **reg_read_condition** (*Any*)
- **reg_read_endness** (*Any*)
- **reg_write_offset** (*Any*)
- **reg_write_expr** (*Any*)
- **reg_write_length** (*Any*)
- **reg_write_condition** (*Any*)
- **reg_write_endness** (*Any*)

- **tmp_read_num** (*Any*)
- **tmp_read_expr** (*Any*)
- **tmp_write_num** (*Any*)
- **tmp_write_expr** (*Any*)
- **expr** (*Any*)
- **expr_result** (*Any*)
- **statement** (*Any*)
- **instruction** (*Any*)
- **address** (*Any*)
- **added_constraints** (*Any*)
- **function_address** (*Any*)
- **exit_target** (*Any*)
- **exit_guard** (*Any*)
- **exit_jumpkind** (*Any*)
- **backtrace** (*Any*)
- **symbolic_name** (*Any*)
- **symbolic_size** (*Any*)
- **symbolic_expr** (*Any*)
- **address_concretization_strategy** (*Any*)
- **address_concretization_action** (*Any*)
- **address_concretization_memory** (*Any*)
- **address_concretization_expr** (*Any*)
- **address_concretization_result** (*Any*)
- **address_concretization_add_constraints** (*Any*)
- **syscall_name** (*Any*)
- **simprocedure_name** (*Any*)
- **simprocedure_addr** (*Any*)
- **simprocedure_result** (*Any*)
- **simprocedure** (*Any*)
- **dirty_name** (*Any*)
- **dirty_handler** (*Any*)
- **dirty_args** (*Any*)
- **dirty_result** (*Any*)
- **sim_engine** (*Any*)
- **sim_successors** (*Any*)
- **mapped_page** (*Any*)

- **mapped_address** (*Any*)

**Return type**
None

class angr.state_plugins.**PTChunk**

Bases: *Chunk*

A chunk, inspired by the implementation of chunks in ptmalloc. Provides a representation of a chunk via a view into the memory plugin. For the chunk definitions and docs that this was loosely based off of, see glibc malloc/malloc.c, line 1033, as of commit 5a580643111ef6081be7b4c7bd1997a5447c903f. Alternatively, take the following link. https://sourceware.org/git/?p=glibc.git;a=blob;f=malloc/malloc.c;h=67cdfd0ad2f003964cd0f7dfe3bcd85ca98528a7;hb=5a580643111ef6081be7b4c7bd1997a5447c903f#l1033

**Variables**

- **base** – the location of the base of the chunk in memory
- **state** – the program state that the chunk is resident in
- **heap** – the heap plugin that the chunk is managed by

**__init__**(*base*, *sim_state*, *heap=None*)

**set_size**(*size*, *is_free=None*)

Use this to set the size on a chunk. When the chunk is new (such as when a free chunk is shrunk to form an allocated chunk and a remainder free chunk) it is recommended that the is_free hint be used since setting the size depends on the chunk's freeness, and vice versa.

**Parameters**

- **size** – size of the chunk
- **is_free** – boolean indicating the chunk's freeness

**set_prev_freeness**(*is_free*)

Sets (or unsets) the flag controlling whether the previous chunk is free.

**Parameters**
**is_free** – if True, sets the previous chunk to be free; if False, sets it to be allocated

**is_prev_free**()

Returns a concrete state of the flag indicating whether the previous chunk is free or not. Issues a warning if that flag is symbolic and has multiple solutions, and then assumes that the previous chunk is free.

**Returns**
True if the previous chunk is free; False otherwise

**prev_size**()

Returns the size of the previous chunk, masking off what would be the flag bits if it were in the actual size field. Performs NO CHECKING to determine whether the previous chunk size is valid (for example, when the previous chunk is not free, its size cannot be determined).

**next_chunk**()

Returns the chunk immediately following (and adjacent to) this one, if it exists.

**Returns**
The following chunk, or None if applicable

**prev_chunk**()

Returns the chunk immediately prior (and adjacent) to this one, if that chunk is free. If the prior chunk is not free, then its base cannot be located and this method raises an error.

**Returns**
If possible, the previous chunk; otherwise, raises an error

**fwd_chunk**()

Returns the chunk following this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the forward chunk; otherwise, raises an error

**bck_chunk**()

Returns the chunk backward from this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the backward chunk; otherwise, raises an error

**class** angr.state_plugins.**PTChunkIterator**

Bases: object

**__init__**(*chunk*, *cond=<function PTChunkIterator.<lambda>>*)

**class** angr.state_plugins.**PosixDevFS**

Bases: *SimMount*

**class** angr.state_plugins.**PosixProcFS**

Bases: *SimMount*

The virtual file system mounted at /proc (as of now, on Linux).

**class** angr.state_plugins.**SimAction**

Bases: *SimEvent*

A SimAction represents a semantic action that an analyzed program performs.

**TMP = 'tmp'**

**REG = 'reg'**

**MEM = 'mem'**

**__init__**(*state*, *region_type*)

Initializes the SimAction.

**Parameters**
**state** – the state that's the SimAction is taking place in.

**property all_objects**

**property is_symbolic**

**property tmp_deps**

**property reg_deps**

**copy**()

**downsize**()

Clears some low-level details (that take up memory) out of the SimAction.

class angr.state_plugins.**SimActionConstraint**

Bases: *SimAction*

A constraint action represents an extra constraint added during execution of a path.

**property all_objects**

**property is_symbolic**

class angr.state_plugins.**SimActionData**

Bases: *SimAction*

A Data action represents a read or a write from memory, registers or a file.

**READ = 'read'**

**WRITE = 'write'**

**OPERATE = 'operate'**

**property all_objects**

**property is_symbolic**

**property tmp_deps**

**property reg_deps**

**property storage**

class angr.state_plugins.**SimActionExit**

Bases: *SimAction*

An Exit action represents a (possibly conditional) jump.

**CONDITIONAL = 'conditional'**

**DEFAULT = 'default'**

**property all_objects**

**property is_symbolic**

class angr.state_plugins.**SimActionObject**

Bases: object

A SimActionObject tracks an AST and its dependencies.

**__init__**(*ast*, *reg_deps=frozenset({})*, *tmp_deps=frozenset({})*, *deps=frozenset({})*, *state=None*)

**Parameters**

- **ast** (*Base*)
- **reg_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **tmp_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **deps** (*frozenset*)
- **state** (*SimState | None*)

**ast: Base**

reg_deps: frozenset[*SimActionData* | *SimActionOperation*]

tmp_deps: frozenset[*SimActionData* | *SimActionOperation*]

to_claripy()

**Return type**
Base

copy()

**Return type**
*SimActionObject*

is_leaf()

**Return type**
bool

property op: str

property args: tuple[ArgType, ...]

property length: int | None

property variables: frozenset[str]

property symbolic: bool

property annotations: tuple[Annotation, ...]

property depth: int

SDiv(*other*)

**Return type**
*SimActionObject*

SMod(*other*)

**Return type**
*SimActionObject*

union(*other*)

**Return type**
*SimActionObject*

intersection(*other*)

**Return type**
*SimActionObject*

widen(*other*)

**Return type**
*SimActionObject*

raw_to_bv()

**Return type**
*SimActionObject*

**bv_to_fp**()

**Return type**
*SimActionObject*

class angr.state_plugins.**SimActionOperation**

Bases: *SimAction*

An action representing an operation between variables and/or constants.

**property all_objects**

**property is_symbolic**

class angr.state_plugins.**SimDebugVariable**

Bases: object

A SimDebugVariable will get dynamically created when queriyng for variable in a state with the SimDebugVariablePlugin. It features a link to the state, an address and a type.

**__init__**(*state*, *addr*, *var_type*)

**Parameters**

- **state** (SimState)
- **var_type** (*VariableType*)

**static from_cle_variable**(*state*, *cle_variable*, *dwarf_cfa*)

**Return type**
*SimDebugVariable*

**Parameters**

- **state** (SimState)
- **cle_variable** (*Variable*)

**property mem_untyped:** ***SimMemView***

**property mem:** ***SimMemView***

**property string:** ***SimMemView***

**with_type**(*sim_type*)

**Return type**
*SimMemView*

**Parameters**
**sim_type** (SimType)

**property resolvable**

**property resolved**

**property concrete**

**store**(*value*)

**property deref:** ***SimDebugVariable***

**array**(*i*)

**Return type**
*SimDebugVariable*

**member**(*member_name*)

**Return type**
*SimDebugVariable*

**Parameters**
**member_name** (*str*)

**class** angr.state_plugins.**SimDebugVariablePlugin**

Bases: *SimStatePlugin*

This is the plugin you'll use to interact with (global/local) program variables. These variables have a name and a visibility scope which depends on the pc address of the state. With this plugin, you can access/modify the value of such variable or find its memory address. For creating program variables, or for importing them from cle, see the knowledge plugin debug_variables. Run `p.kb.dvars.load_from_dwarf()` before using this plugin.

### Example

```
>>> p = angr.Project("various_variables", load_debug_info=True)
>>> p.kb.dvars.load_from_dwarf()
>>> state =  # navigate to the state you want
>>> state.dvars.get_variable("pointer2").deref.mem
<int (32 bits) <BV32 0x1> at 0x404020>
```

**get_variable**(*var_name*)

Returns the visible variable (if any) with name `var_name` based on the current `state.ip`.

**Return type**
*SimDebugVariable*

**Parameters**
**var_name** (*str*)

**property dwarf_cfa**

Returns the current cfa computation. Set this property to the correct value if needed.

**property dwarf_cfa_approx**

**class** angr.state_plugins.**SimEvent**

Bases: `object`

A SimEvent is a log entry for some notable event during symbolic execution. It logs the location it was generated (ins_addr, bbl_addr, stmt_idx, and sim_procedure) as well as arbitrary tags (objects).

You may also be interested in SimAction, which is a specialization of SimEvent for CPU events.

**__init__**(*state*, *event_type*, ***kwargs*)

**class** angr.state_plugins.**SimFilesystem**

Bases: *SimStatePlugin*

angr's emulated filesystem. Available as state.fs. When constructing, all parameters are optional.

**Parameters**

- **files** – A mapping from filepath to SimFile

- **pathsep** – The character used to separate path elements, default forward slash.
- **cwd** – The path of the current working directory to use
- **mountpoints** – A mapping from filepath to SimMountpoint

**Variables**

- **pathsep** – The current pathsep
- **cwd** – The current working directory, relative to the current root
- **unlinks** – A list of unlink operations, tuples of filename and simfile. Be careful, this list is shallow-copied from successor to successor, so don't mutate anything in it without copying.

**__init__**(*files=None*, *pathsep=None*, *cwd=None*, *mountpoints=None*)

**property unlinks**

**property root**

The current root directory, as an absolute path in the filesystem's original namespace, i.e. the one paths were resolved in before any chroot.

**chdir**(*path*)

Changes the current directory to the given path

**chroot**(*path*)

Changes the root directory to the given path, which is interpreted relative to the current root.

This only ever narrows what is reachable: the new root is resolved inside this filesystem, and paths (including "..") are resolved under it from then on. Contrary to Linux, where the current directory is left where it was and can therefore point outside the new root, the current directory is re-anchored inside the new root, since keeping it outside would defeat the point.

**get**(*path*)

Get a file from the filesystem. Returns a SimFile or None.

**insert**(*path*, *simfile*)

Insert a file into the filesystem. Returns whether the operation was successful.

**delete**(*path*)

Remove a file from the filesystem. Returns whether the operation was successful.

This will add a `fs_unlink` event with the path of the file and also the index into the *unlinks* list.

**mount**(*path*, *mount*)

Add a mountpoint to the filesystem.

**unmount**(*path*)

Remove a mountpoint from the filesystem.

**get_mountpoint**(*path*)

Look up the mountpoint servicing the given path.

**Returns**

A tuple of the mount and a list of path elements traversing from the mountpoint to the specified file. If no mountpoint services the path, the list traverses from the real root instead.

class angr.state_plugins.**SimHeapBase**

Bases: *SimStatePlugin*

This is the base heap class that all heap implementations should subclass. It defines a few handlers for common heap functions (the libc memory management functions). Heap implementations are expected to override these functions regardless of whether they implement the SimHeapLibc interface. For an example, see the SimHeapBrk implementation, which is based on the original libc SimProcedure implementations.

**Variables**

- **heap_base** – the address of the base of the heap in memory
- **heap_size** – the maximum size of the main memory region managed by the heap in memory. Only the part of it that has actually been handed out is mapped; see `_ensure_mapped`.
- **mmap_base** – the address of the region from which large mmap allocations will be made

**__init__**(*heap_base=None*, *heap_size=None*)

class angr.state_plugins.**SimHeapBrk**

Bases: *SimHeapBase*

SimHeapBrk represents a trivial heap implementation based on the Unix *brk* system call. This type of heap stores virtually no metadata, so it is up to the user to determine when it is safe to release memory. This also means that it does not properly support standard heap operations like *realloc*.

This heap implementation is a holdover from before any more proper implementations were modelled. At the time, various libc (or win32) SimProcedures handled the heap in the same way that this plugin does now. To make future heap implementations plug-and-playable, they should implement the necessary logic themselves, and dependent SimProcedures should invoke a method by the same name as theirs (prepended with an underscore) upon the heap plugin. Depending on the heap implementation, if the method is not supported, an error should be raised.

Out of consideration for the original way the heap was handled, this plugin implements functionality for all relevant SimProcedures (even those that would not normally be supported together in a single heap implementation).

**Variables**

**heap_location** – the address of the top of the heap, bounding the allocations made starting from *heap_base*

**__init__**(*heap_base=None*, *heap_size=None*)

**allocate**(*sim_size*)

The actual allocation primitive for this heap implementation. Increases the position of the break to allocate space. Has no guards against the heap growing too large.

**Parameters**

**sim_size** – a size specifying how much to increase the break pointer by

**Returns**

a pointer to the previous break position, above which there is now allocated space

**release**(*sim_size*)

The memory release primitive for this heap implementation. Decreases the position of the break to deallocate space. Guards against releasing beyond the initial heap base.

The mapped extent of the heap is deliberately not shrunk here. Unmapping the released pages would discard their contents, so a release followed by an allocation would hand back blank memory instead of the old bytes, which is a behavior change; and since the extent only ever grows to the high-water mark of the break, keeping it costs nothing beyond what the program already asked for.

**Parameters**
**sim_size** – a size specifying how much to decrease the break pointer by (may be symbolic or not)

class angr.state_plugins.**SimHeapLibc**

Bases: *SimHeapBase*

A class of heap that implements the major libc heap management functions.

**malloc**(*sim_size*)

A somewhat faithful implementation of libc *malloc*.

**Parameters**
**sim_size** – the amount of memory (in bytes) to be allocated

**Returns**
the address of the allocation, or a NULL pointer if the allocation failed

**free**(*ptr*)

A somewhat faithful implementation of libc *free*.

**Parameters**
**ptr** – the location in memory to be freed

**calloc**(*sim_nmemb*, *sim_size*)

A somewhat faithful implementation of libc *calloc*.

**Parameters**

- **sim_nmemb** – the number of elements to allocated
- **sim_size** – the size of each element (in bytes)

**Returns**
the address of the allocation, or a NULL pointer if the allocation failed

**realloc**(*ptr*, *size*)

A somewhat faithful implementation of libc *realloc*.

**Parameters**

- **ptr** – the location in memory to be reallocated
- **size** – the new size desired for the allocation

**Returns**
the address of the allocation, or a NULL pointer if the allocation was freed or if no new allocation was made

class angr.state_plugins.**SimHeapPTMalloc**

Bases: *SimHeapFreelist*

A freelist-style heap implementation inspired by ptmalloc. The chunks used by this heap contain heap metadata in addition to user data. While the real-world ptmalloc is implemented using multiple lists of free chunks (corresponding to their different sizes), this more basic model uses a single list of chunks and searches for free chunks using a first-fit algorithm.

**NOTE:** The plugin must be registered using `register_plugin` with name `heap` in order to function properly.

**Variables**

- **heap_base** – the address of the base of the heap in memory
- **heap_size** – the total size of the main memory region managed by the heap in memory

- **mmap_base** – the address of the region from which large mmap allocations will be made
- **free_head_chunk** – the head of the linked list of free chunks in the heap

**__init__**(*heap_base=None*, *heap_size=None*)

**chunk_from_mem**(*ptr*)

Given a pointer to a user payload, return the base of the chunk associated with that payload (i.e. the chunk pointer). Returns None if ptr is null.

**Parameters**
**ptr** – a pointer to the base of a user payload in the heap

**Returns**
a pointer to the base of the associated heap chunk, or None if ptr is null

**class** angr.state_plugins.**SimHostFilesystem**

Bases: *SimConcreteFilesystem*

Simulated mount that makes some piece from the host filesystem available to the guest.

**Parameters**

- **host_path** (*str*) – The path on the host to mount
- **pathsep** (*str*) – The host path separator character, default os.path.sep

**__init__**(*host_path=None*, ***kwargs*)

**class** angr.state_plugins.**SimInspector**

Bases: *SimStatePlugin*

The breakpoint interface, used to instrument execution. For usage information, look here: https://docs.angr.io/core-concepts/simulation#breakpoints

**BP_AFTER = 'after'**

**BP_BEFORE = 'before'**

**BP_BOTH = 'both'**

**__init__**()

**action**(*event_type*, *when*, ***kwargs*)

Called from within the engine when events happens. This function checks all breakpoints registered for that event and fires the ones whose conditions match.

**Return type**
None

**Parameters**

- **event_type** (EventType)
- **when** (When)
- **kwargs** (*dict[str, Any]*)

**make_breakpoint**(*event_type*, *when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

Creates and adds a breakpoint which would trigger on *event_type*. Additional arguments are passed to the BP constructor.

**Returns**
The created breakpoint, so that it can be removed later.

**Parameters**

- **event_type** (EventType)
- **when** (When)
- **enabled** (*bool*)
- **condition** (*Callable[[SimState], bool] | None*)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

**b**(*event_type*, *when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

Creates and adds a breakpoint which would trigger on *event_type*. Additional arguments are passed to the BP constructor.

**Returns**
The created breakpoint, so that it can be removed later.

**Parameters**

- **event_type** (EventType)
- **when** (When)
- **enabled** (*bool*)
- **condition** (*Callable[[SimState], bool] | None*)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

**add_breakpoint**(*event_type*, *bp*)

Adds a breakpoint which would trigger on *event_type*.

**Parameters**

- **event_type** (*EventType*) – The event type to trigger on
- **bp** (*BP*) – The breakpoint

**Return type**
None

**Returns**
The created breakpoint.

**remove_breakpoint**(*event_type*, *bp=None*, *filter_func=None*)

Removes a breakpoint.

**Parameters**

- **bp** (*BP* | None) – The breakpoint to remove.
- **filter_func** (Callable[[*BP*], bool] | None) – A filter function to specify whether each breakpoint should be removed or not.
- **event_type** (EventType)

**Return type**
None

**downsize**()

Reset event-specific attributes on this plugin instance to save memory. This method is supposed to be called by breakpoint implementors. A typical workflow looks like the following:

```
>>> # Add `attr0` and `attr1` via the inspect machinery
>>> self.state.inspect(xxxxxx, attr0=yyyy, attr1=zzzz)
>>> # Get new attributes out of SimInspect in case they are modified by the user
>>> new_attr0 = self.state.inspect.attrs.attr0
>>> new_attr1 = self.state.inspect.attrs.attr1
>>> # Reset them
>>> self.state.inspect.downsize()
```

**class** angr.state_plugins.**SimJavaVmClassloader**

Bases: *SimStatePlugin*

JavaVM Classloader is used as an interface for resolving and initializing Java classes.

**__init__**(*initialized_classes=None*)

**get_class**(*class_name*, *init_class=False*, *step_func=None*)

Get a class descriptor for the class.

**Parameters**

- **class_name** (*str*) – Name of class.
- **init_class** (*bool*) – Whether the class initializer <clinit> should be executed.
- **step_func** (*func*) – Callback function executed at every step of the simulation manager during the execution of the main <clinit> method

**get_superclass**(*class_*)

Get the superclass of the class.

**get_class_hierarchy**(*base_class*)

Walks up the class hierarchy and returns a list of all classes between base class (inclusive) and java.lang.Object (exclusive).

**is_class_initialized**(*class_*)

Indicates whether the classes initializing method <clinit> was already executed on the state.

**init_class**(*class_*, *step_func=None*)

This method simulates the loading of a class by the JVM, during which parts of the class (e.g. static fields) are initialized. For this, we run the class initializer method <clinit> (if available) and update the state accordingly.

**Note: Initialization is skipped, if the class has already been**
initialized (or if it's not loaded in CLE).

**property initialized_classes**

List of all initialized classes.

**class** angr.state_plugins.**SimLightRegisters**

Bases: *SimStatePlugin*

**__init__**(*reg_map=None*, *registers=None*)

**resolve_register**(*offset*, *size*)

**load**(*offset*, *size=None*, ***kwargs*)

**store**(*offset*, *value*, *size=None*, *endness=None*, ***kwargs*)

**class** angr.state_plugins.**SimMemView**

Bases: *SimStatePlugin*

This is a convenient interface with which you can access a program's memory.

The interface works like this:

- You first use [array index notation] to specify the address you'd like to load from
- If at that address is a pointer, you may access the `deref` property to return a SimMemView at the address present in memory.
- You then specify a type for the data by simply accessing a property of that name. For a list of supported types, look at `state.mem.types`.
- You can then *refine* the type. Any type may support any refinement it likes. Right now the only refinements supported are that you may access any member of a struct by its member name, and you may index into a string or array to access that element.
- If the address you specified initially points to an array of that type, you can say *.array(n)* to view the data as an array of n elements.
- Finally, extract the structured data with `.resolved` or `.concrete`. `.resolved` will return bitvector values, while `.concrete` will return integer, string, array, etc values, whatever best represents the data.
- Alternately, you may store a value to memory, by assigning to the chain of properties that you've constructed. Note that because of the way python works, `x = s.mem[...].prop; x = val` will NOT work, you must say `s.mem[...].prop = val`.

For example:

```
>>> s.mem[0x601048].long
<long (64 bits) <BV64 0x4008d0> at 0x601048>
>>> s.mem[0x601048].long.resolved
<BV64 0x4008d0>
>>> s.mem[0x601048].deref
<<untyped> <unresolvable> at 0x4008d0>
>>> s.mem[0x601048].deref.string.concrete
'SOSNEAKY'
```

**__init__**(*ty=None*, *addr=None*, *state=None*)

types: ClassVar[dict] = {'CharT': char, 'DIR': struct DIR, 'FILE': struct FILE, 'FILE_t': struct FILE_t, '_Bool': bool, '_ENTRY': struct _ENTRY, '_IO_codecvt': struct _IO_codecvt, '_IO_iconv_t': struct _IO_iconv_t, '_IO_lock_t': struct pthread_mutex_t, '_IO_marker': struct _IO_marker, '_IO_wide_data': struct _IO_wide_data, '__action_fn_t': __action_fn_t, '__clock_t': uint32_t, '__dev_t': uint64_t, '__free_fn_t': __free_fn_t, '__ftw_func_t': __ftw_func_t, '__gid_t': unsigned int, '__ino64_t': unsigned long long, '__ino_t': unsigned long, '__int128': int128_t, '__int256': int256_t, '__int32': int, '__int64': long long, '__mbstate_t': struct __mbstate_t, '__mode_t': unsigned int, '__nlink_t': unsigned int, '__off64_t': long long, '__off_t': long, '__pid_t': int, '__suseconds_t': int64_t, '__time_t': long, '__uid_t': unsigned int, '_obstack_chunk': struct _obstack_chunk, 'aiocb': struct aiocb, 'aiocb64': struct aiocb64, 'aioinit': struct aioinit, 'argp': struct argp, 'argp_child': struct argp_child, 'argp_option': struct argp_option, 'argp_parser_t': (int, char *, struct argp_state*) -> int, 'argp_state': struct argp_state, 'basic_string': string_t, 'bool': bool, 'byte': uint8_t, 'cc_t': char, 'char': char, 'clock_t': uint32_t, 'comparison_fn_t': comparison_fn_t, 'crypt_data': struct crypt_data, 'dev_t': int, 'dirent': struct dirent, 'dirent64': struct dirent64, 'double': double, 'drand48_data': struct <anon>, 'dword': uint32_t, 'error_t': int, 'exit_status': struct exit_status, 'fd_set': struct fd_set, 'float': float, 'fpos64_t': struct fpos64_t, 'fpos_t': struct fpos_t, 'fstab': struct fstab, 'glob64_t': struct glob64_t, 'glob_t': struct glob_t, 'group': struct group, 'hostent': struct hostent, 'hsearch_data': struct hsearch_data, 'if_nameindex': struct if_nameindex, 'in_addr': struct in_addr, 'in_port_t': uint16_t, 'ino64_t': unsigned long long, 'ino_t': unsigned long, 'int': int, 'int16_t': int16_t, 'int32_t': int32_t, 'int64_t': int64_t, 'int8_t': int8_t, 'iovec': struct <anon>, 'itimerval': struct itimerval, 'lconv': struct lconv, 'long': long, 'long double': double, 'long int': long, 'long long': long long, 'long long int': long long, 'long signed': long, 'long unsigned int': unsigned long, 'mallinfo': struct mallinfo, 'mallinfo2': struct mallinfo2, 'mbstate_t': struct mbstate_t, 'mntent': struct mntent, 'mode_t': unsigned int, 'netent': struct netent, 'ntptimeval': struct ntptimeval, 'obstack': struct obstack, 'off64_t': long long, 'off_t': long, 'option': struct option, 'passwd': struct passwd, 'pid_t': int, 'printf_info': struct printf_info, 'protoent': struct protoent, 'ptrdiff_t': long, 'qword': uint64_t, 'random_data': struct <anon>, 'regex_t': struct regex_t, 'rlim64_t': uint64_t, 'rlim_t': unsigned long, 'rlimit': struct rlimit, 'rlimit64': struct rlimit64, 'rusage': struct rusage, 'sa_family_t': unsigned short, 'sched_param': struct sched_param, 'sem_t': int, 'sembuf': struct sembuf, 'servent': struct servent, 'sgttyb': struct sgttyb, 'short': short, 'short int': short, 'sigevent': struct sigevent, 'sighandler_t': sighandler_t, 'signed': int, 'signed char': char, 'signed int': int, 'signed long': long, 'signed long int': long, 'signed long long': long long, 'signed long long int': long long, 'signed short': short, 'signed short int': short, 'sigset_t': int, 'sigstack': struct sigstack, 'sigval': union sigval { sival_int int; sival_ptr void *; }, 'size_t': size_t, 'sockaddr': struct sockaddr, 'sockaddr_in': struct sockaddr_in, 'socklen_t': uint32_t, 'speed_t': long, 'ssize': size_t, 'ssize_t': size_t, 'std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>': string_t, 'string': string_t, 'struct iovec': struct <anon>, 'struct stat': struct stat, 'struct stat64': struct stat64, 'struct timespec': struct timespec, 'struct timeval': struct timeval, 'tcflag_t': unsigned long, 'termios': struct termios, 'time_t': long, 'timespec': struct timeval, 'timeval': struct timeval, 'timex': struct timex, 'timezone': struct timezone, 'tm': struct tm, 'tms': struct tms, 'uint16_t': uint16_t, 'uint32_t': uint32_t, 'uint64_t': uint64_t, 'uint8_t': uint8_t, 'uintptr_t': unsigned long, 'unsigned': unsigned int, 'unsigned __int128': uint128_t, 'unsigned __int256': uint256_t, 'unsigned char': char, 'unsigned int': unsigned int, 'unsigned long': unsigned long, 'unsigned long int': unsigned long, 'unsigned long long': unsigned long long, 'unsigned long long int': unsigned long long, 'unsigned short': unsigned short, 'unsigned short int': unsigned short, 'utimbuf': struct utimbuf, 'utmp': struct utmp, 'utmpx': struct utmx, 'utsname': struct utsname, 'va_list': struct va_list[1], 'void': void,

**state:** *SimState*[Any, Any] = None

**struct:** *StructMode*

**with_type**(*sim_type*)

Returns a copy of the SimMemView with a type.

**Parameters**
**sim_type** (*SimType*) – The new type.

**Return type**
*SimMemView*

**Returns**
The typed SimMemView copy.

**property resolvable**

**property resolved**

**property concrete**

**property deref:** *SimMemView*

**array**(*n*)

**Return type**
*SimMemView*

**member**(*member_name*)

If self is a struct and member_name is a member of the struct, return that member element. Otherwise raise an exception.

**Return type**
*SimMemView*

**Parameters**
**member_name** (*str*)

**store**(*value*)

**class** angr.state_plugins.**SimMount**

Bases: *SimStatePlugin*

This is the base class for "mount points" in angr's simulated filesystem. Subclass this class and give it to the filesystem to intercept all file creations and opens below the mountpoint. Since this a SimStatePlugin you may also want to implement set_state, copy, merge, etc.

**get**(*path_elements*)

Implement this function to instrument file lookups.

**Parameters**
**path_elements** – A list of path elements traversing from the mountpoint to the file

**Returns**
A SimFile, or None

**insert**(*path_elements*, *simfile*)

Implement this function to instrument file creation.

**Parameters**

- **path_elements** – A list of path elements traversing from the mountpoint to the file
- **simfile** – The file to insert

**Returns**

A bool indicating whether the insert occurred

**delete**(*path_elements*)

Implement this function to instrument file deletion.

**Parameters**

**path_elements** – A list of path elements traversing from the mountpoint to the file

**Returns**

A bool indicating whether the delete occurred

**lookup**(*sim_file*)

Look up the path of a SimFile in the mountpoint

**Parameters**

**sim_file** – A SimFile object needs to be looked up

**Returns**

A string representing the path of the file in the mountpoint Or None if the SimFile does not exist in the mountpoint

**class** angr.state_plugins.**SimRegNameView**

Bases: *SimStatePlugin*

**get**(*reg_name*)

**class** angr.state_plugins.**SimSolver**

Bases: *SimStatePlugin*

This is the plugin you'll use to interact with symbolic variables, creating them and evaluating them. It should be available on a state as `state.solver`.

Any top-level variable of the claripy module can be accessed as a property of this object.

**__init__**(*solver=None*, *all_variables=None*, *temporal_tracked_variables=None*, *eternal_tracked_variables=None*)

**reload_solver**(*constraints=None*)

Reloads the solver. Useful when changing solver options.

**Parameters**

**constraints** (*list*) – A new list of constraints to use in the reloaded solver instead of the current one

**get_variables**(**keys*)

Iterate over all variables for which their tracking key is a prefix of the values provided.

Elements are a tuple, the first element is the full tracking key, the second is the symbol.

```
>>> list(s.solver.get_variables('mem'))
[(('mem', 0x1000), <BV64 mem_1000_4_64>), (('mem', 0x1008), <BV64 mem_1008_5_64>
↪)]
```

```
>>> list(s.solver.get_variables('file'))
[(('file', 1, 0), <BV8 file_1_0_6_8>), (('file', 1, 1), <BV8 file_1_1_7_8>),
    (('file', 2, 0), <BV8 file_2_0_8_8>)]
```

```
>>> list(s.solver.get_variables('file', 2))
[(('file', 2, 0), <BV8 file_2_0_8_8>)]
```

```
>>> list(s.solver.get_variables())
[(('mem', 0x1000), <BV64 mem_1000_4_64>), (('mem', 0x1008), <BV64 mem_1008_5_64>
↪),
    (('file', 1, 0), <BV8 file_1_0_6_8>), (('file', 1, 1), <BV8 file_1_1_7_8>),
    (('file', 2, 0), <BV8 file_2_0_8_8>)]
```

**register_variable**(*v*, *key*, *eternal=True*)

Register a value with the variable tracking system

**Parameters**

- **v** – The BVS to register
- **key** – A tuple to register the variable under

**Parma eternal**
Whether this is an eternal variable, default True. If False, an incrementing counter will be appended to the key.

**describe_variables**(*v*)

Given an AST, iterate over all the keys of all the BVS leaves in the tree which are registered.

**Unconstrained**(*name*, *bits*, *uninitialized=True*, *inspect=True*, *events=True*, *key=None*, *eternal=False*, *uc_alloc_depth=None*, ***kwargs*)

Creates an unconstrained symbol or a default concrete value (0), based on the state options.

**Parameters**

- **name** – The name of the symbol.
- **bits** – The size (in bits) of the symbol.
- **uninitialized** – Whether this value should be counted as an "uninitialized" value in the course of an analysis.
- **inspect** – Set to False to avoid firing SimInspect breakpoints
- **events** – Set to False to avoid generating a SimEvent for the occasion
- **key** – Set this to a tuple of increasingly specific identifiers (for example, `('mem', 0xffbeff00)` or `('file', 4, 0x20)` to cause it to be tracked, i.e. accessible through `solver.get_variables`.
- **eternal** – Set to True in conjunction with setting a key to cause all states with the same ancestry to retrieve the same symbol when trying to create the value. If False, a counter will be appended to the key.

**Returns**
an unconstrained symbol (or a concrete value of 0).

**BVS**(*name*, *size*, *min=None*, *max=None*, *stride=None*, *uninitialized=False*, *explicit_name=False*, *key=None*, *eternal=False*, *inspect=True*, *events=True*, ***kwargs*)

Creates a bit-vector symbol (i.e., a variable). Other keyword parameters are passed directly on to the constructor of claripy.ast.BV.

**Parameters**

- **name** – The name of the symbol.
- **size** – The size (in bits) of the bit-vector.
- **min** – The minimum value of the symbol. Note that this **only** work when using VSA.
- **max** – The maximum value of the symbol. Note that this **only** work when using VSA.
- **stride** – The stride of the symbol. Note that this **only** work when using VSA.
- **uninitialized** – Whether this value should be counted as an "uninitialized" value in the course of an analysis.
- **explicit_name** – Set to True to prevent an identifier from appended to the name to ensure uniqueness.
- **key** – Set this to a tuple of increasingly specific identifiers (for example, `('mem', 0xffbeff00)` or `('file', 4, 0x20)` to cause it to be tracked, i.e. accessible through `solver.get_variables`.
- **eternal** – Set to True in conjunction with setting a key to cause all states with the same ancestry to retrieve the same symbol when trying to create the value. If False, a counter will be appended to the key.
- **inspect** – Set to False to avoid firing SimInspect breakpoints
- **events** – Set to False to avoid generating a SimEvent for the occasion

**Returns**
A BV object representing this symbol.

`downsize()`

Frees memory associated with the constraint solver by clearing all of its internal caches.

`property constraints`

Returns the constraints of the state stored by the solver.

`eval_to_ast(`*e*, *n*, *extra_constraints=()*, *exact=None*`)`

Evaluate an expression, using the solver if necessary. Returns AST objects.

**Parameters**

- **e** – the expression
- **n** – the number of desired solutions
- **extra_constraints** – extra constraints to apply to the solver
- **exact** – if False, returns approximate solutions

**Returns**
a tuple of the solutions, in the form of claripy AST nodes

**Return type**
tuple

**max**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the maximum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the maximum possible value of e (backend object)

**min**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the minimum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the minimum possible value of e (backend object)

**solution**(*e*, *v*, *extra_constraints=()*, *exact=None*)

Return True if *v* is a solution of *expr* with the extra constraints, False otherwise.

**Parameters**

- **e** – An expression (an AST) to evaluate
- **v** – The proposed solution (an AST)
- **extra_constraints** – Extra constraints (as ASTs) to add to the solver for this solve.
- **exact** – If False, return approximate solutions.

**Returns**
True if *v* is a solution of *expr*, False otherwise

**is_true**(*e*, *extra_constraints=()*, *exact=None*)

If the expression provided is absolutely, definitely a true boolean, return True. Note that returning False doesn't necessarily mean that the expression can be false, just that we couldn't figure that out easily.

**Parameters**

- **e** – An expression (an AST) to evaluate
- **extra_constraints** – Extra constraints (as ASTs) to add to the solver for this solve.
- **exact** – If False, return approximate solutions.

**Returns**
True if *v* is definitely true, False otherwise

**is_false**(*e*, *extra_constraints=()*, *exact=None*)

If the expression provided is absolutely, definitely a false boolean, return True. Note that returning False doesn't necessarily mean that the expression can be true, just that we couldn't figure that out easily.

**Parameters**

- **e** – An expression (an AST) to evaluate
- **extra_constraints** – Extra constraints (as ASTs) to add to the solver for this solve.
- **exact** – If False, return approximate solutions.

**Returns**
True if *v* is definitely false, False otherwise

`unsat_core`(*extra_constraints=()*)

This function returns the unsat core from the backend solver.

**Parameters**
`extra_constraints` – Extra constraints (as ASTs) to add to the solver for this solve.

**Returns**
The unsat core.

`satisfiable`(*extra_constraints=()*, *exact=None*)

This function does a constraint check and checks if the solver is in a sat state.

**Parameters**

- `extra_constraints` – Extra constraints (as ASTs) to add to s for this solve
- `exact` – If False, return approximate solutions.

**Returns**
True if sat, otherwise false

`add`(**constraints*)

Add some constraints to the solver.

**Parameters**
`constraints` – Pass any constraints that you want to add (ASTs) as varargs.

`CastType = ~CastType`

`eval_upto`(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression, using the solver if necessary. Returns primitives as specified by the *cast_to* parameter. Only certain primitives are supported, check the implementation of *_cast_to* to see which ones.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- `e` – the expression
- `n` – the number of desired solutions
- `extra_constraints` – extra constraints to apply to the solver
- `exact` – if False, returns approximate solutions
- `cast_to` – desired type of resulting values

**Returns**
a tuple of the solutions, in the form of Python primitives

**Return type**
tuple

`eval`(*e*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get any possible solution. The desired output types can be specified using the *cast_to* parameter. *extra_constraints* can be used to specify additional constraints the returned values must satisfy.

**Overloads**

- **self**, **e** (claripy.ast.BV), **cast_to** (None), **kwargs** → int
- **self**, **e** (claripy.ast.BV), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.Bool), **cast_to** (None), **kwargs** → bool
- **self**, **e** (claripy.ast.Bool), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.FP), **cast_to** (None), **kwargs** → float
- **self**, **e** (claripy.ast.FP), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.String | str), **cast_to** (None), **kwargs** → str
- **self**, **e** (_EvalArg), **cast_to** (None), **kwargs** → Any
- **self**, **e** (_EvalArg), **cast_to** (type[CastType]), **kwargs** → CastType

**Parameters**

- **e** – the expression to get a solution for
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*
- **cast_to** – desired type of resulting values

**Raises**
*SimUnsatError* – if no solution could be found satisfying the given constraints

**Returns**

`eval_one`(*e*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get the only possible solution. Errors if either no or more than one solution is returned. A kwarg parameter *default* can be specified to be returned instead of failure!

**Overloads**

- **self**, **e** (claripy.ast.BV), **cast_to** (None), **kwargs** → int
- **self**, **e** (claripy.ast.BV), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.Bool), **cast_to** (None), **kwargs** → bool
- **self**, **e** (claripy.ast.Bool), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.FP), **cast_to** (None), **kwargs** → float
- **self**, **e** (claripy.ast.FP), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.String | str), **cast_to** (None), **kwargs** → str
- **self**, **e** (_EvalArg), **cast_to** (None), **kwargs** → Any
- **self**, **e** (_EvalArg), **cast_to** (type[CastType]), **kwargs** → CastType

**Parameters**

- **e** – the expression to get a solution for

- **cast_to** – desired type of resulting values
- **default** – A value can be passed as a kwarg here. It will be returned in case of failure.
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if more than one solution was found to satisfy the given constraints

**Returns**

The value for *e*

**eval_atmost**(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get at most *n* possible solutions. Errors if either none or more than *n* solutions are returned.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- **e** – the expression to get a solution for
- **n** – the inclusive upper limit on the number of solutions
- **cast_to** – desired type of resulting values
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if more than *n* solutions were found to satisfy the given constraints

**Returns**

The solutions for *e*

**eval_atleast**(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get at least *n* possible solutions. Errors if less than *n* solutions were found.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]

- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- **e** – the expression to get a solution for
- **n** – the inclusive lower limit on the number of solutions
- **cast_to** – desired type of resulting values
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if less than *n* solutions were found to satisfy the given constraints

**Returns**

The solutions for *e*

**eval_exact**(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get exactly the *n* possible solutions. Errors if any number of solutions other than *n* was found to exist.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- **e** – the expression to get a solution for
- **n** – the inclusive lower limit on the number of solutions
- **cast_to** – desired type of resulting values
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints

- *SimValueError* – if any number of solutions other than *n* were found to satisfy the given constraints

**Returns**
The solutions for *e*

**min_int**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the minimum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the minimum possible value of e (backend object)

**max_int**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the maximum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the maximum possible value of e (backend object)

**unique**(*e*, ***kwargs*)

Returns True if the expression *e* has only one solution by querying the constraint solver. It does also add that unique solution to the solver's constraints.

**symbolic**(*e*)

Returns True if the expression *e* is symbolic.

**single_valued**(*e*)

Returns True whether *e* is a concrete value or is a value set with only 1 possible value. This differs from *unique* in that this *does* not query the constraint solver.

**simplify**(*e=None*)

Simplifies *e*. If *e* is None, simplifies the constraints of this state.

**variables**(*e*)

Returns the symbolic variables present in the AST of *e*.

**class** angr.state_plugins.**SimStateCGC**

Bases: *SimStatePlugin*

This state plugin keeps track of CGC state.

**EBADF = 1**

**EFAULT = 2**

**EINVAL = 3**

**ENOMEM = 4**

**ENOSYS = 5**

**EPIPE = 6**

**FD_SETSIZE = 1024**

**max_allocation = 268435456**

**__init__**()

**peek_input**()

**discard_input**(*num_bytes*)

**peek_output**()

**discard_output**(*num_bytes*)

**addr_invalid**(*a*)

**get_max_sinkhole**(*length*)

Find a sinkhole which is large enough to support *length* bytes.

This uses first-fit. The first sinkhole (ordered in descending order by their address) which can hold *length* bytes is chosen. If there are more than *length* bytes in the sinkhole, a new sinkhole is created representing the remaining bytes while the old sinkhole is removed.

**add_sinkhole**(*address*, *length*)

Add a sinkhole.

Allow the possibility for the program to reuse the memory represented by the address length pair.

**class** angr.state_plugins.**SimStateEdgeHitmap**

Bases: *SimStatePlugin*

A state plugin that stores AFL-style edge hitmap coverage data.

This plugin is used by the Icicle engine to track edge coverage during concrete execution. It stores the hitmap as raw bytes, which can be efficiently passed to and from the Icicle emulator.

This plugin is NOT registered as a default plugin - it must be explicitly added to states that need edge coverage tracking.

**HITMAP_SIZE = 65536**

**__init__**(*edge_hitmap=None*)

Initialize the edge hitmap plugin.

**Parameters**

**edge_hitmap** (bytes | None) – Initial edge hitmap bytes, or None to start with a zeroed hitmap.

**copy**(*memo*)

Create a copy of the plugin.

**Parameters**

**memo** – Memoization dictionary to avoid duplicate copies.

**Returns**

A new SimStateEdgeHitmap instance with a copy of the hitmap.

**merge**(*others*, *merge_conditions*, *common_ancestor=None*)

Merge multiple hitmap plugins.

For coverage tracking, we OR the hitmaps together to preserve all observed edges.

**Parameters**

- **others** – Other plugin instances to merge with.
- **merge_conditions** – Symbolic conditions for each merge branch.

- **common_ancestor** – Common ancestor plugin instance.

**Returns**
True if merge was successful.

class angr.state_plugins.**SimStateGlobals**

Bases: *SimStatePlugin*

**__init__**(*backer=None*)

**keys**()

**values**()

**items**()

**get**(*k*, *alt=None*)

**pop**(*k*, *alt=None*)

class angr.state_plugins.**SimStateHistory**

Bases: *SimStatePlugin*

This class keeps track of historically-relevant information for paths.

**__init__**(*parent=None*, *clone=None*)

**property addr**

**trim**()

Discard the ancestry of this state.

**filter_actions**(*start_block_addr=None*, *end_block_addr=None*, *block_stmt=None*, *insn_addr=None*, *read_from=None*, *write_to=None*)

Filter self.actions based on some common parameters.

[start_block_addr, end_block_addr]

**Parameters**

- **start_block_addr** – Only return actions generated in blocks starting at this address.
- **end_block_addr** – Only return actions generated in blocks ending at this address.
- **block_stmt** – Only return actions generated in the nth statement of each block.
- **insn_addr** – Only return actions generated in the assembly instruction at this address.
- **read_from** – Only return actions that perform a read from the specified location.
- **write_to** – Only return actions that perform a write to the specified location.

Notes: If IR optimization is turned on, reads and writes may not occur in the instruction they originally came from. Most commonly, If a register is read from twice in the same block, the second read will not happen, instead reusing the temp the value is already stored in.

Valid values for read_from and write_to are the string literals 'reg' or 'mem' (matching any read or write to registers or memory, respectively), any string (representing a read or write to the named register), and any integer (representing a read or write to the memory at this address).

**demote**()

Demotes this history node, causing it to drop the strong state reference.

reachable()

add_event(*event_type*, ***kwargs*)

add_action(*action*)

extend_actions(*new_actions*)

subscribe_actions()

property recent_constraints

property recent_actions

property block_count

property lineage

property parents

property events: Reversible[*SimEvent*]

property actions: Reversible[*SimAction*]

property jumpkinds: Reversible[str]

property jump_guards: Reversible[Bool]

property jump_targets

property jump_sources

property descriptions: Reversible[str]

property bbl_addrs: Reversible[int]

property ins_addrs: Reversible[int]

property stack_actions

property last_edge_hitmap: *bytes* | None

Returns the last edge hitmap in the history chain, or None if there is no edge hitmap.

closest_common_ancestor(*other*)

Find the common ancestor between this history node and 'other'.

**Parameters**
**other** – the PathHistory to find a common ancestor with.

**Returns**
the common ancestor SimStateHistory, or None if there isn't one

constraints_since(*other*)

Returns the constraints that have been accumulated since *other*.

**Parameters**
**other** – a prior PathHistory object

**Returns**
a list of constraints

**make_child**()

**class** angr.state_plugins.**SimStateIcicle**

Bases: *SimStatePlugin*

Engine-internal plugin for IcicleEngine continuation detection.

Attached to states produced by `IcicleEngine.process()`. Owns the VM and the metadata the engine needs to decide whether the next call is a lightweight continuation or requires a full snapshot restore.

**__init__**(*vm_ref=None*, *generation=None*, *base_translation_data=None*, *translation_data=None*, *dirty_pages=None*)

**Parameters**

- **vm_ref** (*IcicleVMRef | None*)
- **generation** (*int | None*)
- **base_translation_data** (*IcicleStateTranslationData | None*)
- **translation_data** (*IcicleStateTranslationData | None*)
- **dirty_pages** (*set[int] | None*)

**property is_live: bool**

True when the VM is still positioned where this state last left it.

**class** angr.state_plugins.**SimStateJNIReferences**

Bases: *SimStatePlugin*

Management of the mapping between opaque JNI references and the corresponding Java objects.

**__init__**(*local_refs=None*, *global_refs=None*)

**lookup**(*opaque_ref*)

Lookups the object that was used for creating the reference.

**create_new_reference**(*obj*, *global_ref=False*)

Create a new reference thats maps to the given object.

**Parameters**

- **obj** – Object which gets referenced.
- **global_ref** (*bool*) – Whether a local or global reference is created.

**clear_local_references**()

Clear all local references.

**delete_reference**(*opaque_ref*, *global_ref=False*)

Delete the stored mapping of a reference.

**Parameters**

- **opaque_ref** – Reference which should be removed.
- **global_ref** (*bool*) – Whether opaque_ref is a local or global reference.

**class** angr.state_plugins.**SimStateLibc**

Bases: *SimStatePlugin*

This state plugin keeps track of various libc stuff:

```
LOCALE_ARRAY = [b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x03
', b'\x02 ', b'\x02 ', b'\x02 ', b'\x02 ', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x01`', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8',
b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x08\xd6', b'\x08\xd6', b'\x08\xd6', b'\x08\xd6',
b'\x08\xd6', b'\x08\xd6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x02\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
```

TOLOWER_LOC_ARRAY = [128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 4294967295, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 97, 98,
99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
116, 117, 118, 119, 120, 121, 122, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101,
102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186,
187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203,
204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237,
238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
255]

TOUPPER_LOC_ARRAY = [128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 4294967295, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 123, 124, 125, 126, 127,
128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178,
179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212,
213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229,
230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
247, 248, 249, 250, 251, 252, 253, 254, 255]

__init__()

property errno

ret_errno(*val*)

class angr.state_plugins.**SimStateLog**

Bases: *SimStatePlugin*

**__init__**(*log=None*)

**property actions**

**add_event**(*event_type*, ***kwargs*)

**add_action**(*action*)

**extend_actions**(*new_actions*)

**events_of_type**(*event_type*)

**actions_of_type**(*action_type*)

**property fresh_constraints**

**clear**()

class angr.state_plugins.**SimStateLoopData**

Bases: *SimStatePlugin*

This class keeps track of loop-related information for states. Note that we have 2 counters for loop iterations (trip counts): the first recording the number of times one of the back edges (or continue edges) of a loop is taken, whereas the second recording the number of times the loop header (or loop entry) is executed. These 2 counters may differ since compilers usually optimize loops hence completely change the loop structure at the binary level. This is supposed to be used with *LoopSeer* exploration technique, which monitors loop execution. For the moment, the only thing we want to analyze is loop trip counts, but nothing prevents us from extending this plugin for other loop analyses.

**__init__**(*back_edge_trip_counts=None*, *header_trip_counts=None*, *current_loop=None*)

**Parameters**

- **back_edge_trip_counts** – Dictionary that stores back edge based trip counts for each loop. Keys are address of loop headers.
- **header_trip_counts** – Dictionary that stores header based trip counts for each loop. Keys are address of loop headers.
- **current_loop** – List of currently running loops. Each element is a tuple (loop object, list of loop exits).

class angr.state_plugins.**SimStatePlugin**

Bases: object

This is a base class for SimState plugins. A SimState plugin will be copied along with the state when the state is branched. They are intended to be used for things such as tracking open files, tracking heap details, and providing storage and persistence for SimProcedures.

**__init__**()

**Return type**
None

**state:** *SimState*[Any, Any]

**set_state**(*state*)

Sets a new state (for example, if the state has been branched)

**Return type**
None

**static memo**(*f*)

A decorator function you should apply to copy

**Return type**
_CopyFunc[TypeVar(S_co)]

**Parameters**
**f** (*Callable[[Any, dict[int, Any]], S_co]*)

**copy**(*_memo*)

Should return a copy of the plugin without any state attached. Should check the memo first, and add itself to memo if it ends up making a new copy.

In order to simplify using the memo, you should annotate implementations of this function with SimStatePlugin.memo

The base implementation of this function constructs a new instance of the plugin's class without calling its initializer. If you super-call down to it, make sure you instantiate all the fields in your copy method!

**Parameters**

- **memo** – A dictionary mapping object identifiers (id(obj)) to their copied instance. Use this to avoid infinite recursion and diverged copies.
- **_memo** (*dict[int, Any]*)

**Return type**
*SimStatePlugin*

**merge**(*others*, *merge_conditions*, *common_ancestor=None*)

Should merge the state plugin with the provided others. This will be called by state.merge() after copying the target state, so this should mutate the current instance to merge with the others.

Note that when multiple instances of a single plugin object (for example, a file) are referenced in the state, it is important that merge only ever be called once. This should be solved by designating one of the plugin's referees as the "real owner", who should be the one to actually merge it. This technique doesn't work to resolve the similar issue that arises during copying because merging doesn't produce a new reference to insert.

There will be n others and n+1 merge conditions, since the first condition corresponds to self. To match elements up to conditions, say zip([self] + others, merge_conditions)

When implementing this, make sure that you "deepen" both others and common_ancestor before calling sub-elements' merge methods, e.g.

```
self.foo.merge(
    [o.foo for o in others],
    merge_conditions,
    common_ancestor=common_ancestor.foo if common_ancestor is not None else None
)
```

During static analysis, merge_conditions can be None, in which case you should use state.solver.union(values). TODO: fish please make this less bullshit

There is a utility `claripy.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = claripy.ite_cases(zip(conditions[1:], [o.bar for o in others]), self.bar)`

**Parameters**

- **others** – the other state plugins to merge with
- **merge_conditions** – a symbolic condition for each of the plugins
- **common_ancestor** – a common ancestor of this plugin and the others being merged

**Returns**
True if the state plugins are actually merged.

**Return type**
bool

**classmethod register_default**(*name*, *xtr=None*)

**Return type**
None

**Parameters**

- **name** (*str*)
- **xtr** (type[SimStatePlugin] | *str* | *None*)

**init_state**()

Use this function to perform any initialization on the state at plugin-add time

**Return type**
None

**class** angr.state_plugins.**SimStatePreconstrainer**

Bases: *SimStatePlugin*

This state plugin manages the concept of preconstraining - adding constraints which you would like to remove later.

**Parameters**
**constrained_addrs** – SimActions for memory operations whose addresses should be constrained during crash analysis

**__init__**(*constrained_addrs=None*)

**preconstrain**(*value*, *variable*)

Add a preconstraint that `variable == value` to the state.

**Parameters**

- **value** – The concrete value. Can be a bitvector or a bytestring or an integer.
- **variable** – The BVS to preconstrain.

**preconstrain_file**(*content*, *simfile*, *set_length=False*)

Preconstrain the contents of a file.

**Parameters**

- **content** – The content to preconstrain the file to. Can be a bytestring or a list thereof.
- **simfile** – The actual simfile to preconstrain

**preconstrain_flag_page**(*magic_content*)

Preconstrain the data in the flag page.

**Parameters**
**magic_content** – The content of the magic page as a bytestring.

**remove_preconstraints**(*to_composite_solver=True*, *simplify=True*)

Remove the preconstraints from the state.

If you are using the zen plugin, this will also use that to filter the constraints.

**Parameters**

- **to_composite_solver** – Whether to convert the replacement solver to a composite solver. You probably want this if you're switching from tracing to symbolic analysis.
- **simplify** – Whether to simplify the resulting set of constraints.

**reconstrain**()

Split the solver. If any of the subsolvers time out after a short timeout (10 seconds), re-add the preconstraints associated with each of its variables. Hopefully these constraints still allow us to do meaningful things to the state.

**class** angr.state_plugins.**SimStateScratch**

Bases: *SimStatePlugin*

Implements the scratch state plugin.

**__init__**(*scratch=None*)

**property ail_block_idx**

**property is_ail**

**property priv**

**push_priv**(*priv*)

**pop_priv**()

**set_tyenv**(*tyenv*)

**tmp_expr**(*tmp*)

Returns the Claripy expression of a VEX temp value.

**Parameters**

- **tmp** – the number of the tmp
- **simplify** – simplify the tmp before returning it

**Returns**
a Claripy expression of the tmp

**store_tmp**(*tmp*, *content*, *reg_deps=frozenset({})*, *tmp_deps=frozenset({})*, *deps=None*, ***kwargs*)

Stores a Claripy expression in a VEX temp value. If in symbolic mode, this involves adding a constraint for the tmp's symbolic variable.

**Parameters**

- **tmp** – the number of the tmp
- **content** – a Claripy expression of the content

- **reg_deps** – the register dependencies of the content
- **tmp_deps** – the temporary value dependencies of the content

**clear**()

**class** angr.state_plugins.**SimSymbolizer**

Bases: *SimStatePlugin*

The symbolizer state plugin ensures that pointers that are stored in memory are symbolic. This allows for the tracking of and reasoning over these pointers (for example, to reason about memory disclosure).

**__init__**()

**set_symbolization_for_all_pages**()

Sets the symbolizer to symbolize pointers to all pages as they are written to memory..

**set_symbolized_target_range**(*base*, *length*)

All pointers to the target range will be symbolized as they are written to memory.

Due to optimizations, the _pages_ containing this range will be set as symbolization targets, not just the range itself.

**resymbolize**()

Re-symbolizes all pointers in memory. This can be called to symbolize any pointers to target regions that were written (and not mangled beyond recognition) before symbolization was set.

**class** angr.state_plugins.**SimSystemPosix**

Bases: *SimStatePlugin*

Data storage and interaction mechanisms for states with an environment conforming to posix. Available as `state.posix`.

**SIG_BLOCK = 0**

**SIG_UNBLOCK = 1**

**SIG_SETMASK = 2**

**EPERM = 1**

**ENOENT = 2**

**ESRCH = 3**

**EINTR = 4**

**EIO = 5**

**ENXIO = 6**

**E2BIG = 7**

**ENOEXEC = 8**

**EBADF = 9**

**ECHILD = 10**

**EAGAIN = 11**

**ENOMEM = 12**

**EACCES = 13**

**EFAULT = 14**

**ENOTBLK = 15**

**EBUSY = 16**

**EEXIST = 17**

**EXDEV = 18**

**ENODEV = 19**

**ENOTDIR = 20**

**EISDIR = 21**

**EINVAL = 22**

**ENFILE = 23**

**EMFILE = 24**

**ENOTTY = 25**

**ETXTBSY = 26**

**EFBIG = 27**

**ENOSPC = 28**

**ESPIPE = 29**

**EROFS = 30**

**EMLINK = 31**

**EPIPE = 32**

**EDOM = 33**

**ERANGE = 34**

**__init__**(*stdin=None*, *stdout=None*, *stderr=None*, *fd=None*, *sockets=None*, *socket_queue=None*, *argv=None*, *argc=None*, *environ=None*, *auxv=None*, *tls_modules=None*, *sigmask=None*, *pid=None*, *ppid=None*, *uid=None*, *gid=None*, *brk=None*)

**property closed_fds**

**set_brk**(*new_brk*)

**open**(*name*, *flags*, *preferred_fd=None*)

Open a symbolic file. Basically open(2).

**Parameters**

- **name** (`string or bytes`) – Path of the symbolic file, as a string or bytes.
- **flags** – File operation flags, a bitfield of constants from open(2), as an AST

- **preferred_fd** – Assign this fd if it's not already claimed.

**Returns**
The file descriptor number allocated (maps through posix.get_fd to a SimFileDescriptor) or -1 if the open fails.

mode from open(2) is unsupported at present.

**open_socket**(*ident*)

**get_fd**(*fd*, *create_file=True*)

Looks up the SimFileDescriptor associated with the given number (an AST). If the number is concrete and does not map to anything, return None. If the number is symbolic, constrain it to an open fd and create a new file for it. Set create_file to False if no write-access is planned (i.e. fd is read-only).

**get_concrete_fd**(*fd*, *create_file=True*)

Same behavior as get_fd(fd), only the result is a concrete integer fd (or -1) instead of a SimFileDescriptor.

**close**(*fd*)

Closes the given file descriptor (an AST). Returns whether the operation succeeded (a concrete boolean)

**fstat**(*fd*)

**fstat_with_result**(*sim_fd*)

**sigmask**(*sigsetsize=None*)

Gets the current sigmask. If it's blank, a new one is created (of sigsetsize).

**Parameters**
**sigsetsize** – the size (in *bytes* of the sigmask set)

**Returns**
the sigmask

**sigprocmask**(*how*, *new_mask*, *sigsetsize*, *valid_ptr=True*)

Updates the signal mask.

**Parameters**

- **how** – the "how" argument of sigprocmask (see manpage)
- **new_mask** – the mask modification to apply
- **sigsetsize** – the size (in *bytes* of the sigmask set)
- **valid_ptr** – is set if the new_mask was not NULL

**dump_file_by_path**(*path*, ***kwargs*)

Returns the concrete content for a file by path.

**Parameters**

- **path** – file path as string
- **kwargs** – passed to state.solver.eval

**Returns**
file contents as string

**dumps**(*fd*, ***kwargs*)

Returns the concrete content for a file descriptor.

BACKWARD COMPATIBILITY: if you ask for file descriptors 0 1 or 2, it will return the data from stdin, stdout, or stderr as a flat string.

**Parameters**
**fd** – A file descriptor.

**Returns**
The concrete content.

**Return type**
str

class angr.state_plugins.**SimUCManager**

Bases: *SimStatePlugin*

**__init__**(*man=None*)

**assign**(*dst_addr_ast*)

Assign a new region for under-constrained symbolic execution.

**Parameters**
**dst_addr_ast** – the symbolic AST which address of the new allocated region will be assigned to.

**Returns**
as ast of memory address that points to a new region

**get_alloc_depth**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*int | Base*)

**set_alloc_depth**(*addr*, *depth*)

**Parameters**

- **addr** (*Base*)
- **depth** (*int*)

**is_bounded**(*ast*)

Test whether an AST is bounded by any existing constraint in the related solver.

**Parameters**
**ast** – an claripy.AST object

**Returns**
True if there is at least one related constraint, False otherwise

class angr.state_plugins.**Stat**

Bases: tuple

Stat(st_dev, st_ino, st_nlink, st_mode, st_uid, st_gid, st_rdev, st_size, st_blksize, st_blocks, st_atime, st_atimensec, st_mtime, st_mtimensec, st_ctime, st_ctimensec)

**static __new__**(*_cls*, *st_dev*, *st_ino*, *st_nlink*, *st_mode*, *st_uid*, *st_gid*, *st_rdev*, *st_size*, *st_blksize*, *st_blocks*, *st_atime*, *st_atimensec*, *st_mtime*, *st_mtimensec*, *st_ctime*, *st_ctimensec*)

Create new instance of Stat(st_dev, st_ino, st_nlink, st_mode, st_uid, st_gid, st_rdev, st_size, st_blksize, st_blocks, st_atime, st_atimensec, st_mtime, st_mtimensec, st_ctime, st_ctimensec)

**st_atime**

Alias for field number 10

**st_atimensec**
Alias for field number 11

**st_blksize**
Alias for field number 8

**st_blocks**
Alias for field number 9

**st_ctime**
Alias for field number 14

**st_ctimensec**
Alias for field number 15

**st_dev**
Alias for field number 0

**st_gid**
Alias for field number 5

**st_ino**
Alias for field number 1

**st_mode**
Alias for field number 3

**st_mtime**
Alias for field number 12

**st_mtimensec**
Alias for field number 13

**st_nlink**
Alias for field number 2

**st_rdev**
Alias for field number 6

**st_size**
Alias for field number 7

**st_uid**
Alias for field number 4

**class** angr.state_plugins.**StructMode**
Bases: object

**__init__**(*view*)

**class** angr.state_plugins.**Unicorn**
Bases: *SimStatePlugin*

setup the unicorn engine for a state

**UC_CONFIG = {}**

**__init__**(*syscall_hooks=None*, *cache_key=None*, *unicount=None*, *symbolic_var_counts=None*, *symbolic_inst_counts=None*, *concretized_asts=None*, *always_concretize=None*, *never_concretize=None*, *concretize_at=None*, *concretization_threshold_memory=None*, *concretization_threshold_registers=None*, *concretization_threshold_instruction=None*, *cooldown_symbolic_stop=2*, *cooldown_unsupported_stop=2*, *cooldown_nonunicorn_blocks=100*, *cooldown_stop_point=1*, *max_steps=1000000*)

Initializes the Unicorn plugin for angr. This plugin handles communication with UnicornEngine.

**property uc**

**static delete_uc**()

**set_last_block_details**(*details*)

**set_stops**(*stop_points*)

**set_tracking**(*track_bbls*, *track_stack*)

**hook**()

**uncache_region**(*addr*, *length*)

**clear_page_cache**()

**setup**(*syscall_data=None*, *fd_bytes=None*)

**start**(*step=None*)

**get_recent_bbl_addrs**()

**get_stop_details**()

**finish**(*succ_state*)

**destroy**(*succ_state*)

**set_regs**()

setting unicorn registers

**setup_flags**()

**setup_gdt**(*fs*, *gs*)

**read_msr**(*msr=3221225728*)

**write_msr**(*val*, *msr=3221225728*)

**get_regs**(*succ_state*)

loading registers from unicorn. If succ_state is not None, update it instead of self.state. Needed when handling symbolic exits in native interface

angr.state_plugins.**resource_event**(*state*, *exception*)

continues on next page

Table 108 – continued from previous page

### Submodules

| | |
|---|---|
| *callstack* | |
| *cgc* | |
| *debug_variables* | |
| *edge_hitmap* | |
| *filesystem* | |
| *gdb* | |
| *globals* | |
| *heap* | |
| *history* | |
| *icicle* | |
| *inspect* | |
| *javavm_classloader* | |
| *jni_references* | |
| *libc* | |
| *light_registers* | |
| *log* | |
| *loop_data* | |
| *plugin* | |
| *posix* | |
| *preconstrainer* | |
| *scratch* | |
| *sim_action* | |
| *sim_action_object* | |
| *sim_event* | |
| *solver* | |
| *symbolizer* | |
| *trace_additions* | This file contains objects to track additional information during a trace or modify symbolic variables during a trace. |
| *uc_manager* | |
| *unicorn_engine* | |
| *view* | |

## 10.43.1 angr.state_plugins.callstack

**class** angr.state_plugins.callstack.**CallStack**

Bases: *SimStatePlugin*

Stores the address of the function you're in and the value of SP at the VERY BOTTOM of the stack, i.e. points to the return address.

**__init__**(*call_site_addr=0*, *func_addr=0*, *stack_ptr=0*, *ret_addr=0*, *jumpkind='Ijk_Call'*, *next_frame=None*, *invoke_return_variable=None*)

**Parameters**
**next_frame** (*Self | None*)

**copy_without_tail**(*memo*)

**property current_function_address**

Address of the current function.

**Returns**
the address of the function

**Return type**
int

**property current_stack_pointer**

Get the value of the stack pointer.

**Returns**
Value of the stack pointer

**Return type**
int

**property current_return_target**

Get the return target.

**Returns**
The address of return target.

**Return type**
int

**static stack_suffix_to_string**(*stack_suffix*)

Convert a stack suffix to a human-readable string representation. :type tuple stack_suffix: :param tuple stack_suffix: The stack suffix. :return: A string representation :rtype: str

**property top: Self**

Returns the element at the top of the callstack without removing it.

**Returns**
A CallStack.

**push**(*cf*)

Push the frame cf onto the stack. Return the new stack.

**Return type**
Self

**Parameters**
**cf** (*Self*)

**pop**()

Pop the top frame from the stack. Return the new stack.

**call**(*callsite_addr*, *addr*, *retn_target=None*, *stack_pointer=None*)

Push a stack frame into the call stack. This method is called when calling a function in CFG recovery.

**Parameters**

- **callsite_addr** (*int*) – Address of the call site
- **addr** (*int*) – Address of the call target
- **retn_target** (*int or None*) – Address of the return target
- **stack_pointer** (*int*) – Value of the stack pointer

**Returns**
None

**ret**(*retn_target=None*)

Pop one or many call frames from the stack. This method is called when returning from a function in CFG recovery.

**Parameters**
**retn_target** (*int*) – The target to return to.

**Returns**
None

**dbg_repr**()

Debugging representation of this CallStack object.

**Returns**
Details of this CalLStack

**Return type**
str

**stack_suffix**(*context_sensitivity_level*)

Generate the stack suffix. A stack suffix can be used as the key to a SimRun in CFG recovery.

**Parameters**
**context_sensitivity_level** (*int*) – Level of context sensitivity.

**Returns**
A tuple of stack suffix.

**Return type**
tuple[int | None, ...]

**class** angr.state_plugins.callstack.**CallStackAction**

Bases: object

Used in callstack backtrace, which is a history of callstacks along a path, to record individual actions occurred each time the callstack is changed.

**__init__**(*callstack_hash*, *callstack_depth*, *action*, *callframe=None*, *ret_site_addr=None*)

### 10.43.2 angr.state_plugins.cgc

**class** angr.state_plugins.cgc.**SimStateCGC**

Bases: *SimStatePlugin*

This state plugin keeps track of CGC state.

**EBADF = 1**

**EFAULT = 2**

**EINVAL = 3**

**ENOMEM = 4**

**ENOSYS = 5**

**EPIPE = 6**

**FD_SETSIZE = 1024**

**max_allocation = 268435456**

**__init__**()

**peek_input**()

**discard_input**(*num_bytes*)

**peek_output**()

**discard_output**(*num_bytes*)

**addr_invalid**(*a*)

**get_max_sinkhole**(*length*)

Find a sinkhole which is large enough to support *length* bytes.

This uses first-fit. The first sinkhole (ordered in descending order by their address) which can hold *length* bytes is chosen. If there are more than *length* bytes in the sinkhole, a new sinkhole is created representing the remaining bytes while the old sinkhole is removed.

**add_sinkhole**(*address*, *length*)

Add a sinkhole.

Allow the possibility for the program to reuse the memory represented by the address length pair.

### 10.43.3 angr.state_plugins.debug_variables

**class** angr.state_plugins.debug_variables.**SimDebugVariable**

Bases: object

A SimDebugVariable will get dynamically created when queriyng for variable in a state with the SimDebugVariablePlugin. It features a link to the state, an address and a type.

**__init__**(*state*, *addr*, *var_type*)

**Parameters**

- **state** (SimState)
- **var_type** (*VariableType*)

**static from_cle_variable**(*state*, *cle_variable*, *dwarf_cfa*)

**Return type**
*SimDebugVariable*

**Parameters**

- **state** (SimState)
- **cle_variable** (*Variable*)

**property mem_untyped:** *SimMemView*

**property mem:** *SimMemView*

**property string:** *SimMemView*

**with_type**(*sim_type*)

**Return type**
*SimMemView*

**Parameters**
**sim_type** (SimType)

**property resolvable**

**property resolved**

**property concrete**

**store**(*value*)

**property deref:** *SimDebugVariable*

**array**(*i*)

**Return type**
*SimDebugVariable*

**member**(*member_name*)

**Return type**
*SimDebugVariable*

**Parameters**
**member_name** (*str*)

**class** angr.state_plugins.debug_variables.**SimDebugVariablePlugin**

Bases: *SimStatePlugin*

This is the plugin you'll use to interact with (global/local) program variables. These variables have a name and a visibility scope which depends on the pc address of the state. With this plugin, you can access/modify the value of such variable or find its memory address. For creating program variables, or for importing them from cle, see the knowledge plugin debug_variables. Run `p.kb.dvars.load_from_dwarf()` before using this plugin.

### Example

```
>>> p = angr.Project("various_variables", load_debug_info=True)
>>> p.kb.dvars.load_from_dwarf()
>>> state =  # navigate to the state you want
>>> state.dvars.get_variable("pointer2").deref.mem
<int (32 bits) <BV32 0x1> at 0x404020>
```

**get_variable**(*var_name*)

Returns the visible variable (if any) with name `var_name` based on the current `state.ip`.

**Return type**
*SimDebugVariable*

**Parameters**
**var_name** (*str*)

**property dwarf_cfa**

Returns the current cfa computation. Set this property to the correct value if needed.

**property dwarf_cfa_approx**

### 10.43.4 angr.state_plugins.edge_hitmap

**class** angr.state_plugins.edge_hitmap.**SimStateEdgeHitmap**

Bases: *SimStatePlugin*

A state plugin that stores AFL-style edge hitmap coverage data.

This plugin is used by the Icicle engine to track edge coverage during concrete execution. It stores the hitmap as raw bytes, which can be efficiently passed to and from the Icicle emulator.

This plugin is NOT registered as a default plugin - it must be explicitly added to states that need edge coverage tracking.

**HITMAP_SIZE = 65536**

**__init__**(*edge_hitmap=None*)

Initialize the edge hitmap plugin.

**Parameters**
**edge_hitmap** (bytes | None) – Initial edge hitmap bytes, or None to start with a zeroed hitmap.

**edge_hitmap:** *bytes* **| None**

**copy**(*memo*)

Create a copy of the plugin.

**Parameters**
**memo** – Memoization dictionary to avoid duplicate copies.

**Returns**
A new SimStateEdgeHitmap instance with a copy of the hitmap.

**merge**(*others*, *merge_conditions*, *common_ancestor=None*)

Merge multiple hitmap plugins.

For coverage tracking, we OR the hitmaps together to preserve all observed edges.

**Parameters**

- **others** – Other plugin instances to merge with.
- **merge_conditions** – Symbolic conditions for each merge branch.
- **common_ancestor** – Common ancestor plugin instance.

**Returns**
True if merge was successful.

### 10.43.5 angr.state_plugins.filesystem

**class** angr.state_plugins.filesystem.**Stat**

Bases: tuple

Stat(st_dev, st_ino, st_nlink, st_mode, st_uid, st_gid, st_rdev, st_size, st_blksize, st_blocks, st_atime, st_atimensec, st_mtime, st_mtimensec, st_ctime, st_ctimensec)

**static __new__**(*_cls*, *st_dev*, *st_ino*, *st_nlink*, *st_mode*, *st_uid*, *st_gid*, *st_rdev*, *st_size*, *st_blksize*, *st_blocks*, *st_atime*, *st_atimensec*, *st_mtime*, *st_mtimensec*, *st_ctime*, *st_ctimensec*)

Create new instance of Stat(st_dev, st_ino, st_nlink, st_mode, st_uid, st_gid, st_rdev, st_size, st_blksize, st_blocks, st_atime, st_atimensec, st_mtime, st_mtimensec, st_ctime, st_ctimensec)

**st_atime**
Alias for field number 10

**st_atimensec**
Alias for field number 11

**st_blksize**
Alias for field number 8

**st_blocks**
Alias for field number 9

**st_ctime**
Alias for field number 14

**st_ctimensec**
Alias for field number 15

**st_dev**
Alias for field number 0

**st_gid**
Alias for field number 5

**st_ino**
Alias for field number 1

**st_mode**
Alias for field number 3

**st_mtime**
Alias for field number 12

**st_mtimensec**
Alias for field number 13

**st_nlink**
Alias for field number 2

**st_rdev**
Alias for field number 6

**st_size**
Alias for field number 7

**st_uid**
Alias for field number 4

**class** angr.state_plugins.filesystem.**SimFilesystem**

Bases: *SimStatePlugin*

angr's emulated filesystem. Available as state.fs. When constructing, all parameters are optional.

**Parameters**

- **files** – A mapping from filepath to SimFile
- **pathsep** – The character used to separate path elements, default forward slash.
- **cwd** – The path of the current working directory to use

- **mountpoints** – A mapping from filepath to SimMountpoint

**Variables**

- **pathsep** – The current pathsep
- **cwd** – The current working directory, relative to the current root
- **unlinks** – A list of unlink operations, tuples of filename and simfile. Be careful, this list is shallow-copied from successor to successor, so don't mutate anything in it without copying.

**__init__**(*files=None*, *pathsep=None*, *cwd=None*, *mountpoints=None*)

**property unlinks**

**property root**

The current root directory, as an absolute path in the filesystem's original namespace, i.e. the one paths were resolved in before any chroot.

**chdir**(*path*)

Changes the current directory to the given path

**chroot**(*path*)

Changes the root directory to the given path, which is interpreted relative to the current root.

This only ever narrows what is reachable: the new root is resolved inside this filesystem, and paths (including "..") are resolved under it from then on. Contrary to Linux, where the current directory is left where it was and can therefore point outside the new root, the current directory is re-anchored inside the new root, since keeping it outside would defeat the point.

**get**(*path*)

Get a file from the filesystem. Returns a SimFile or None.

**insert**(*path*, *simfile*)

Insert a file into the filesystem. Returns whether the operation was successful.

**delete**(*path*)

Remove a file from the filesystem. Returns whether the operation was successful.

This will add a `fs_unlink` event with the path of the file and also the index into the *unlinks* list.

**mount**(*path*, *mount*)

Add a mountpoint to the filesystem.

**unmount**(*path*)

Remove a mountpoint from the filesystem.

**get_mountpoint**(*path*)

Look up the mountpoint servicing the given path.

**Returns**

A tuple of the mount and a list of path elements traversing from the mountpoint to the specified file. If no mountpoint services the path, the list traverses from the real root instead.

**class** angr.state_plugins.filesystem.**SimMount**

Bases: *SimStatePlugin*

This is the base class for "mount points" in angr's simulated filesystem. Subclass this class and give it to the filesystem to intercept all file creations and opens below the mountpoint. Since this a SimStatePlugin you may also want to implement set_state, copy, merge, etc.

**get**(*path_elements*)

Implement this function to instrument file lookups.

**Parameters**
**path_elements** – A list of path elements traversing from the mountpoint to the file

**Returns**
A SimFile, or None

**insert**(*path_elements*, *simfile*)

Implement this function to instrument file creation.

**Parameters**

- **path_elements** – A list of path elements traversing from the mountpoint to the file
- **simfile** – The file to insert

**Returns**
A bool indicating whether the insert occurred

**delete**(*path_elements*)

Implement this function to instrument file deletion.

**Parameters**
**path_elements** – A list of path elements traversing from the mountpoint to the file

**Returns**
A bool indicating whether the delete occurred

**lookup**(*sim_file*)

Look up the path of a SimFile in the mountpoint

**Parameters**
**sim_file** – A SimFile object needs to be looked up

**Returns**
A string representing the path of the file in the mountpoint Or None if the SimFile does not exist in the mountpoint

**class** angr.state_plugins.filesystem.**SimConcreteFilesystem**

Bases: *SimMount*

Abstract SimMount allowing the user to import files from some external source into the guest

**Parameters**
**pathsep** (*str*) – The host path separator character, default os.path.sep

**__init__**(*pathsep='/'*)

**class** angr.state_plugins.filesystem.**SimHostFilesystem**

Bases: *SimConcreteFilesystem*

Simulated mount that makes some piece from the host filesystem available to the guest.

**Parameters**

- **host_path** (*str*) – The path on the host to mount
- **pathsep** (*str*) – The host path separator character, default os.path.sep

**__init__**(*host_path=None*, ***kwargs*)

### 10.43.6 angr.state_plugins.gdb

**class** angr.state_plugins.gdb.**GDB**

Bases: *SimStatePlugin*

Initialize or update a state from gdb dumps of the stack, heap, registers and data (or arbitrary) segments.

**__init__**(*omit_fp=False*, *adjust_stack=False*)

**Parameters**

- **omit_fp** – The frame pointer register is used for something else. (i.e. –omit_frame_pointer)
- **adjust_stack** – Use different stack addresses than the gdb session (not recommended).

**set_stack**(*stack_dump*, *stack_top*)

Stack dump is a dump of the stack from gdb, i.e. the result of the following gdb command :

`dump binary memory [stack_dump] [begin_addr] [end_addr]`

We set the stack to the same addresses as the gdb session to avoid pointers corruption.

**Parameters**

- **stack_dump** – The dump file.
- **stack_top** – The address of the top of the stack in the gdb session.

**set_heap**(*heap_dump*, *heap_base*)

Heap dump is a dump of the heap from gdb, i.e. the result of the following gdb command:

`dump binary memory [stack_dump] [begin] [end]`

**Parameters**

- **heap_dump** – The dump file.
- **heap_base** – The start address of the heap in the gdb session.

**set_data**(*addr*, *data_dump*)

Update any data range (most likely use is the data segments of loaded objects)

**set_regs**(*regs_dump*)

Initialize register values within the state

**Parameters**

**regs_dump** – The output of `info registers` in gdb.

### 10.43.7 angr.state_plugins.globals

**class** angr.state_plugins.globals.**SimStateGlobals**

Bases: *SimStatePlugin*

**__init__**(*backer=None*)

**keys**()

**values**()

**items**()

**get**(*k*, *alt=None*)

**pop**(*k*, *alt=None*)

### 10.43.8 angr.state_plugins.heap

**class** angr.state_plugins.heap.**PTChunk**

Bases: *Chunk*

A chunk, inspired by the implementation of chunks in ptmalloc. Provides a representation of a chunk via a view into the memory plugin. For the chunk definitions and docs that this was loosely based off of, see glibc malloc/malloc.c, line 1033, as of commit 5a580643111ef6081be7b4c7bd1997a5447c903f. Alternatively, take the following link. https://sourceware.org/git/?p=glibc.git;a=blob;f=malloc/malloc.c;h=67cdfd0ad2f003964cd0f7dfe3bcd85ca98528a7;hb=5a580643111ef6081be7b4c7bd1997a5447c903f#l1033

**Variables**

- **base** – the location of the base of the chunk in memory
- **state** – the program state that the chunk is resident in
- **heap** – the heap plugin that the chunk is managed by

**__init__**(*base*, *sim_state*, *heap=None*)

**set_size**(*size*, *is_free=None*)

Use this to set the size on a chunk. When the chunk is new (such as when a free chunk is shrunk to form an allocated chunk and a remainder free chunk) it is recommended that the is_free hint be used since setting the size depends on the chunk's freeness, and vice versa.

**Parameters**

- **size** – size of the chunk
- **is_free** – boolean indicating the chunk's freeness

**set_prev_freeness**(*is_free*)

Sets (or unsets) the flag controlling whether the previous chunk is free.

**Parameters**

**is_free** – if True, sets the previous chunk to be free; if False, sets it to be allocated

**is_prev_free**()

Returns a concrete state of the flag indicating whether the previous chunk is free or not. Issues a warning if that flag is symbolic and has multiple solutions, and then assumes that the previous chunk is free.

**Returns**

True if the previous chunk is free; False otherwise

**prev_size**()

Returns the size of the previous chunk, masking off what would be the flag bits if it were in the actual size field. Performs NO CHECKING to determine whether the previous chunk size is valid (for example, when the previous chunk is not free, its size cannot be determined).

**next_chunk**()

Returns the chunk immediately following (and adjacent to) this one, if it exists.

**Returns**

The following chunk, or None if applicable

**prev_chunk**()

Returns the chunk immediately prior (and adjacent) to this one, if that chunk is free. If the prior chunk is not free, then its base cannot be located and this method raises an error.

**Returns**
If possible, the previous chunk; otherwise, raises an error

**fwd_chunk**()

Returns the chunk following this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the forward chunk; otherwise, raises an error

**bck_chunk**()

Returns the chunk backward from this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the backward chunk; otherwise, raises an error

**class** angr.state_plugins.heap.**PTChunkIterator**

Bases: object

**__init__**(*chunk*, *cond=<function PTChunkIterator.<lambda>>*)

**class** angr.state_plugins.heap.**SimHeapBase**

Bases: *SimStatePlugin*

This is the base heap class that all heap implementations should subclass. It defines a few handlers for common heap functions (the libc memory management functions). Heap implementations are expected to override these functions regardless of whether they implement the SimHeapLibc interface. For an example, see the SimHeapBrk implementation, which is based on the original libc SimProcedure implementations.

**Variables**

- **heap_base** – the address of the base of the heap in memory
- **heap_size** – the maximum size of the main memory region managed by the heap in memory. Only the part of it that has actually been handed out is mapped; see _ensure_mapped.
- **mmap_base** – the address of the region from which large mmap allocations will be made

**__init__**(*heap_base=None*, *heap_size=None*)

**class** angr.state_plugins.heap.**SimHeapBrk**

Bases: *SimHeapBase*

SimHeapBrk represents a trivial heap implementation based on the Unix *brk* system call. This type of heap stores virtually no metadata, so it is up to the user to determine when it is safe to release memory. This also means that it does not properly support standard heap operations like *realloc*.

This heap implementation is a holdover from before any more proper implementations were modelled. At the time, various libc (or win32) SimProcedures handled the heap in the same way that this plugin does now. To make future heap implementations plug-and-playable, they should implement the necessary logic themselves, and dependent SimProcedures should invoke a method by the same name as theirs (prepended with an underscore) upon the heap plugin. Depending on the heap implementation, if the method is not supported, an error should be raised.

Out of consideration for the original way the heap was handled, this plugin implements functionality for all relevant SimProcedures (even those that would not normally be supported together in a single heap implementation).

**Variables**
**heap_location** – the address of the top of the heap, bounding the allocations made starting from *heap_base*

**__init__**(*heap_base=None*, *heap_size=None*)

**allocate**(*sim_size*)

The actual allocation primitive for this heap implementation. Increases the position of the break to allocate space. Has no guards against the heap growing too large.

**Parameters**
**sim_size** – a size specifying how much to increase the break pointer by

**Returns**
a pointer to the previous break position, above which there is now allocated space

**release**(*sim_size*)

The memory release primitive for this heap implementation. Decreases the position of the break to deallocate space. Guards against releasing beyond the initial heap base.

The mapped extent of the heap is deliberately not shrunk here. Unmapping the released pages would discard their contents, so a release followed by an allocation would hand back blank memory instead of the old bytes, which is a behavior change; and since the extent only ever grows to the high-water mark of the break, keeping it costs nothing beyond what the program already asked for.

**Parameters**
**sim_size** – a size specifying how much to decrease the break pointer by (may be symbolic or not)

**class** angr.state_plugins.heap.**SimHeapLibc**

Bases: *SimHeapBase*

A class of heap that implements the major libc heap management functions.

**malloc**(*sim_size*)

A somewhat faithful implementation of libc *malloc*.

**Parameters**
**sim_size** – the amount of memory (in bytes) to be allocated

**Returns**
the address of the allocation, or a NULL pointer if the allocation failed

**free**(*ptr*)

A somewhat faithful implementation of libc *free*.

**Parameters**
**ptr** – the location in memory to be freed

**calloc**(*sim_nmemb*, *sim_size*)

A somewhat faithful implementation of libc *calloc*.

**Parameters**

- **sim_nmemb** – the number of elements to allocated
- **sim_size** – the size of each element (in bytes)

**Returns**
the address of the allocation, or a NULL pointer if the allocation failed

**realloc**(*ptr*, *size*)

A somewhat faithful implementation of libc *realloc*.

**Parameters**

- **ptr** – the location in memory to be reallocated
- **size** – the new size desired for the allocation

**Returns**

the address of the allocation, or a NULL pointer if the allocation was freed or if no new allocation was made

**class** angr.state_plugins.heap.**SimHeapPTMalloc**

Bases: *SimHeapFreelist*

A freelist-style heap implementation inspired by ptmalloc. The chunks used by this heap contain heap metadata in addition to user data. While the real-world ptmalloc is implemented using multiple lists of free chunks (corresponding to their different sizes), this more basic model uses a single list of chunks and searches for free chunks using a first-fit algorithm.

**NOTE:** The plugin must be registered using `register_plugin` with name `heap` in order to function properly.

**Variables**

- **heap_base** – the address of the base of the heap in memory
- **heap_size** – the total size of the main memory region managed by the heap in memory
- **mmap_base** – the address of the region from which large mmap allocations will be made
- **free_head_chunk** – the head of the linked list of free chunks in the heap

**__init__**(*heap_base=None*, *heap_size=None*)

**chunk_from_mem**(*ptr*)

Given a pointer to a user payload, return the base of the chunk associated with that payload (i.e. the chunk pointer). Returns None if ptr is null.

**Parameters**

**ptr** – a pointer to the base of a user payload in the heap

**Returns**

a pointer to the base of the associated heap chunk, or None if ptr is null

## Submodules


*heap_base*
*heap_brk*
*heap_freelist*
*heap_libc*
*heap_ptmalloc*
*utils*


### angr.state_plugins.heap.heap_base

**class** angr.state_plugins.heap.heap_base.**SimHeapBase**

Bases: *SimStatePlugin*

This is the base heap class that all heap implementations should subclass. It defines a few handlers for common heap functions (the libc memory management functions). Heap implementations are expected to override these functions regardless of whether they implement the SimHeapLibc interface. For an example, see the SimHeapBrk implementation, which is based on the original libc SimProcedure implementations.

**Variables**

- **heap_base** – the address of the base of the heap in memory
- **heap_size** – the maximum size of the main memory region managed by the heap in memory. Only the part of it that has actually been handed out is mapped; see `_ensure_mapped`.
- **mmap_base** – the address of the region from which large mmap allocations will be made

**__init__**(*heap_base=None*, *heap_size=None*)

### angr.state_plugins.heap.heap_brk

**class** angr.state_plugins.heap.heap_brk.**SimHeapBrk**

Bases: *SimHeapBase*

SimHeapBrk represents a trivial heap implementation based on the Unix *brk* system call. This type of heap stores virtually no metadata, so it is up to the user to determine when it is safe to release memory. This also means that it does not properly support standard heap operations like *realloc*.

This heap implementation is a holdover from before any more proper implementations were modelled. At the time, various libc (or win32) SimProcedures handled the heap in the same way that this plugin does now. To make future heap implementations plug-and-playable, they should implement the necessary logic themselves, and dependent SimProcedures should invoke a method by the same name as theirs (prepended with an underscore) upon the heap plugin. Depending on the heap implementation, if the method is not supported, an error should be raised.

Out of consideration for the original way the heap was handled, this plugin implements functionality for all relevant SimProcedures (even those that would not normally be supported together in a single heap implementation).

**Variables**

**heap_location** – the address of the top of the heap, bounding the allocations made starting from *heap_base*

**__init__**(*heap_base=None*, *heap_size=None*)

**allocate**(*sim_size*)

The actual allocation primitive for this heap implementation. Increases the position of the break to allocate space. Has no guards against the heap growing too large.

**Parameters**

**sim_size** – a size specifying how much to increase the break pointer by

**Returns**

a pointer to the previous break position, above which there is now allocated space

**release**(*sim_size*)

The memory release primitive for this heap implementation. Decreases the position of the break to deallocate space. Guards against releasing beyond the initial heap base.

The mapped extent of the heap is deliberately not shrunk here. Unmapping the released pages would discard their contents, so a release followed by an allocation would hand back blank memory instead of the old bytes, which is a behavior change; and since the extent only ever grows to the high-water mark of the break, keeping it costs nothing beyond what the program already asked for.

**Parameters**
**sim_size** – a size specifying how much to decrease the break pointer by (may be symbolic or not)

### angr.state_plugins.heap.heap_freelist

**class** angr.state_plugins.heap.heap_freelist.**Chunk**

Bases: object

The sort of chunk as would typically be found in a freelist-style heap implementation. Provides a representation of a chunk via a view into the memory plugin. Chunks may be adjacent, in different senses, to as many as four other chunks. For any given chunk, two of these chunks are adjacent to it in memory, and are referred to as the "previous" and "next" chunks throughout this implementation. For any given free chunk, there may also be two significant chunks that are adjacent to it in some linked list of free chunks. These chunks are referred to the "backward" and "forward" chunks relative to the chunk in question.

**Variables**

- **base** – the location of the base of the chunk in memory
- **state** – the program state that the chunk is resident in

**__init__**(*base*, *sim_state*)

**get_size**()
Returns the actual size of a chunk (as opposed to the entire size field, which may include some flags).

**get_data_size**()
Returns the size of the data portion of a chunk.

**set_size**(*size*)
Sets the size of the chunk, preserving any flags.

**data_ptr**()
Returns the address of the payload of the chunk.

**is_free**()
Returns a concrete determination as to whether the chunk is free.

**next_chunk**()
Returns the chunk immediately following (and adjacent to) this one.

**prev_chunk**()
Returns the chunk immediately prior (and adjacent) to this one.

**fwd_chunk**()
Returns the chunk following this chunk in the list of free chunks.

**set_fwd_chunk**(*fwd*)
Sets the chunk following this chunk in the list of free chunks.

**Parameters**
**fwd** – the chunk to follow this chunk in the list of free chunks

**bck_chunk**()
Returns the chunk backward from this chunk in the list of free chunks.

**set_bck_chunk**(*bck*)

Sets the chunk backward from this chunk in the list of free chunks.

**Parameters**
**bck** – the chunk to precede this chunk in the list of free chunks

class angr.state_plugins.heap.heap_freelist.**SimHeapFreelist**

Bases: *SimHeapLibc*

A freelist-style heap implementation. Distinguishing features of such heaps include chunks containing heap metadata in addition to user data and at least (but often more than) one linked list of free chunks.

**chunks**()

Returns an iterator over all the chunks in the heap.

**allocated_chunks**()

Returns an iterator over all the allocated chunks in the heap.

**free_chunks**()

Returns an iterator over all the free chunks in the heap.

**chunk_from_mem**(*ptr*)

Given a pointer to a user payload, return the chunk associated with that payload.

**Parameters**
**ptr** – a pointer to the base of a user payload in the heap

**Returns**
the associated heap chunk

**print_heap_state**()

**print_all_chunks**()

### angr.state_plugins.heap.heap_libc

class angr.state_plugins.heap.heap_libc.**SimHeapLibc**

Bases: *SimHeapBase*

A class of heap that implements the major libc heap management functions.

**malloc**(*sim_size*)

A somewhat faithful implementation of libc *malloc*.

**Parameters**
**sim_size** – the amount of memory (in bytes) to be allocated

**Returns**
the address of the allocation, or a NULL pointer if the allocation failed

**free**(*ptr*)

A somewhat faithful implementation of libc *free*.

**Parameters**
**ptr** – the location in memory to be freed

**calloc**(*sim_nmemb*, *sim_size*)

A somewhat faithful implementation of libc *calloc*.

**Parameters**

- **sim_nmemb** – the number of elements to allocated
- **sim_size** – the size of each element (in bytes)

**Returns**
the address of the allocation, or a NULL pointer if the allocation failed

**realloc**(*ptr*, *size*)

A somewhat faithful implementation of libc *realloc*.

**Parameters**

- **ptr** – the location in memory to be reallocated
- **size** – the new size desired for the allocation

**Returns**
the address of the allocation, or a NULL pointer if the allocation was freed or if no new allocation was made

### angr.state_plugins.heap.heap_ptmalloc

**class** angr.state_plugins.heap.heap_ptmalloc.**PTChunk**

Bases: *Chunk*

A chunk, inspired by the implementation of chunks in ptmalloc. Provides a representation of a chunk via a view into the memory plugin. For the chunk definitions and docs that this was loosely based off of, see glibc malloc/malloc.c, line 1033, as of commit 5a580643111ef6081be7b4c7bd1997a5447c903f. Alternatively, take the following link. https://sourceware.org/git/?p=glibc.git;a=blob;f=malloc/malloc.c;h=67cdfd0ad2f003964cd0f7dfe3bcd85ca98528a7;hb=5a580643111ef6081be7b4c7bd1997a5447c903f#l1033

**Variables**

- **base** – the location of the base of the chunk in memory
- **state** – the program state that the chunk is resident in
- **heap** – the heap plugin that the chunk is managed by

**__init__**(*base*, *sim_state*, *heap=None*)

**set_size**(*size*, *is_free=None*)

Use this to set the size on a chunk. When the chunk is new (such as when a free chunk is shrunk to form an allocated chunk and a remainder free chunk) it is recommended that the is_free hint be used since setting the size depends on the chunk's freeness, and vice versa.

**Parameters**

- **size** – size of the chunk
- **is_free** – boolean indicating the chunk's freeness

**set_prev_freeness**(*is_free*)

Sets (or unsets) the flag controlling whether the previous chunk is free.

**Parameters**
**is_free** – if True, sets the previous chunk to be free; if False, sets it to be allocated

**is_prev_free**()

Returns a concrete state of the flag indicating whether the previous chunk is free or not. Issues a warning if that flag is symbolic and has multiple solutions, and then assumes that the previous chunk is free.

**Returns**
True if the previous chunk is free; False otherwise

**prev_size**()

Returns the size of the previous chunk, masking off what would be the flag bits if it were in the actual size field. Performs NO CHECKING to determine whether the previous chunk size is valid (for example, when the previous chunk is not free, its size cannot be determined).

**next_chunk**()

Returns the chunk immediately following (and adjacent to) this one, if it exists.

**Returns**
The following chunk, or None if applicable

**prev_chunk**()

Returns the chunk immediately prior (and adjacent) to this one, if that chunk is free. If the prior chunk is not free, then its base cannot be located and this method raises an error.

**Returns**
If possible, the previous chunk; otherwise, raises an error

**fwd_chunk**()

Returns the chunk following this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the forward chunk; otherwise, raises an error

**bck_chunk**()

Returns the chunk backward from this chunk in the list of free chunks. If this chunk is not free, then it resides in no such list and this method raises an error.

**Returns**
If possible, the backward chunk; otherwise, raises an error

**class** angr.state_plugins.heap.heap_ptmalloc.**PTChunkIterator**

Bases: object

**__init__**(*chunk*, *cond=<function PTChunkIterator.<lambda>>*)

**class** angr.state_plugins.heap.heap_ptmalloc.**SimHeapPTMalloc**

Bases: *SimHeapFreelist*

A freelist-style heap implementation inspired by ptmalloc. The chunks used by this heap contain heap metadata in addition to user data. While the real-world ptmalloc is implemented using multiple lists of free chunks (corresponding to their different sizes), this more basic model uses a single list of chunks and searches for free chunks using a first-fit algorithm.

**NOTE:** The plugin must be registered using `register_plugin` with name `heap` in order to function properly.

**Variables**

- **heap_base** – the address of the base of the heap in memory
- **heap_size** – the total size of the main memory region managed by the heap in memory
- **mmap_base** – the address of the region from which large mmap allocations will be made
- **free_head_chunk** – the head of the linked list of free chunks in the heap

**__init__**(*heap_base=None*, *heap_size=None*)

**chunk_from_mem**(*ptr*)

Given a pointer to a user payload, return the base of the chunk associated with that payload (i.e. the chunk pointer). Returns None if ptr is null.

**Parameters**
**ptr** – a pointer to the base of a user payload in the heap

**Returns**
a pointer to the base of the associated heap chunk, or None if ptr is null

### angr.state_plugins.heap.utils

angr.state_plugins.heap.utils.**concretize**(*x*, *solver*, *sym_handler*)

For now a lot of naive concretization is done when handling heap metadata to keep things manageable. This idiom showed up a lot as a result, so to reduce code repetition this function uses a callback to handle the one or two operations that varied across invocations.

**Parameters**

- **x** – the item to be concretized
- **solver** – the solver to evaluate the item with
- **sym_handler** – the handler to be used when the item may take on more than one value

**Returns**
a concrete value for the item

## 10.43.9 angr.state_plugins.history

**class** angr.state_plugins.history.**SimStateHistory**

Bases: *SimStatePlugin*

This class keeps track of historically-relevant information for paths.

**__init__**(*parent=None*, *clone=None*)

**property addr**

**trim**()

Discard the ancestry of this state.

**filter_actions**(*start_block_addr=None*, *end_block_addr=None*, *block_stmt=None*, *insn_addr=None*, *read_from=None*, *write_to=None*)

Filter self.actions based on some common parameters.

[start_block_addr, end_block_addr]

**Parameters**

- **start_block_addr** – Only return actions generated in blocks starting at this address.
- **end_block_addr** – Only return actions generated in blocks ending at this address.
- **block_stmt** – Only return actions generated in the nth statement of each block.
- **insn_addr** – Only return actions generated in the assembly instruction at this address.
- **read_from** – Only return actions that perform a read from the specified location.
- **write_to** – Only return actions that perform a write to the specified location.

Notes: If IR optimization is turned on, reads and writes may not occur in the instruction they originally came from. Most commonly, If a register is read from twice in the same block, the second read will not happen, instead reusing the temp the value is already stored in.

Valid values for read_from and write_to are the string literals 'reg' or 'mem' (matching any read or write to registers or memory, respectively), any string (representing a read or write to the named register), and any integer (representing a read or write to the memory at this address).

**demote**()

Demotes this history node, causing it to drop the strong state reference.

**reachable**()

**add_event**(*event_type*, ***kwargs*)

**add_action**(*action*)

**extend_actions**(*new_actions*)

**subscribe_actions**()

**property recent_constraints**

**property recent_actions**

**property block_count**

**property lineage**

**property parents**

**property events:** Reversible[*SimEvent*]

**property actions:** Reversible[*SimAction*]

**property jumpkinds:** Reversible[str]

**property jump_guards:** Reversible[Bool]

**property jump_targets**

**property jump_sources**

**property descriptions:** Reversible[str]

**property bbl_addrs:** Reversible[int]

**property ins_addrs:** Reversible[int]

**property stack_actions**

**property last_edge_hitmap:** *bytes* | None

Returns the last edge hitmap in the history chain, or None if there is no edge hitmap.

**closest_common_ancestor**(*other*)

Find the common ancestor between this history node and 'other'.

**Parameters**
**other** – the PathHistory to find a common ancestor with.

**Returns**
the common ancestor SimStateHistory, or None if there isn't one

**constraints_since**(*other*)

Returns the constraints that have been accumulated since *other*.

**Parameters**
**other** – a prior PathHistory object

**Returns**
a list of constraints

**make_child**()

**class** angr.state_plugins.history.**TreeIter**

Bases: object

**__init__**(*start*, *end=None*)

**property hardcopy**

**count**(*v*)

Count occurrences of value v in the entire history. Note that the subclass must implement the __reversed__ method, otherwise an exception will be thrown. :type object v: :param object v: The value to look for :return: The number of occurrences :rtype: int

**class** angr.state_plugins.history.**HistoryIter**

Bases: *TreeIter*

**class** angr.state_plugins.history.**LambdaAttrIter**

Bases: *TreeIter*

**__init__**(*start*, *f*, ***kwargs*)

**class** angr.state_plugins.history.**LambdaIterIter**

Bases: *LambdaAttrIter*

**__init__**(*start*, *f*, *reverse=True*, ***kwargs*)

### 10.43.10 angr.state_plugins.icicle

**class** angr.state_plugins.icicle.**IcicleStateTranslationData**

Bases: object

Represents the saved information needed to convert an Icicle state back to an angr state.

**base_state:** *SimState*[int, int]

**registers:** set[str]

**mapped_pages:** set[int]

**writable_pages:** set[int]

**explicit_page_metadata:** dict[int, int | None]

**initial_cpu_icount:** int

**icicle_arch:** str

__init__(*base_state*, *registers*, *mapped_pages*, *writable_pages*, *explicit_page_metadata*, *initial_cpu_icount*, *icicle_arch*)

**Parameters**

- **base_state** (*SimState[int, int]*)
- **registers** (*set[str]*)
- **mapped_pages** (*set[int]*)
- **writable_pages** (*set[int]*)
- **explicit_page_metadata** (*dict[int, int | None]*)
- **initial_cpu_icount** (*int*)
- **icicle_arch** (*str*)

**Return type**
None

class angr.state_plugins.icicle.**IcicleVMRef**

Bases: object

Holder shared by reference across plugin copies.

Lets multiple SimStateIcicle plugins point at the same VM and observe each other's advancements via *generation*: each successful engine run bumps *generation*, invalidating any plugin still holding the prior value.

**vm: Icicle**

**generation: int = 0**

__init__(*vm*, *generation=0*)

**Parameters**

- **vm** (*Icicle*)
- **generation** (*int*)

**Return type**
None

class angr.state_plugins.icicle.**SimStateIcicle**

Bases: *SimStatePlugin*

Engine-internal plugin for IcicleEngine continuation detection.

Attached to states produced by `IcicleEngine.process()`. Owns the VM and the metadata the engine needs to decide whether the next call is a lightweight continuation or requires a full snapshot restore.

__init__(*vm_ref=None*, *generation=None*, *base_translation_data=None*, *translation_data=None*, *dirty_pages=None*)

**Parameters**

- **vm_ref** (*IcicleVMRef | None*)
- **generation** (*int | None*)
- **base_translation_data** (*IcicleStateTranslationData | None*)
- **translation_data** (*IcicleStateTranslationData | None*)
- **dirty_pages** (*set[int] | None*)

**property is_live:** bool

True when the VM is still positioned where this state last left it.

### 10.43.11 angr.state_plugins.inspect

**class** angr.state_plugins.inspect.**EventType**

Bases: StrEnum

Event types for breakpoints.

**VEX_LIFT = 'vex_lift'**

**MEM_READ = 'mem_read'**

**MEM_WRITE = 'mem_write'**

**ADDRESS_CONCRETIZATION = 'address_concretization'**

**REG_READ = 'reg_read'**

**REG_WRITE = 'reg_write'**

**TMP_READ = 'tmp_read'**

**TMP_WRITE = 'tmp_write'**

**EXPR = 'expr'**

**STATEMENT = 'statement'**

**INSTRUCTION = 'instruction'**

**IRSB = 'irsb'**

**CONSTRAINTS = 'constraints'**

**EXIT = 'exit'**

**FORK = 'fork'**

**SYMBOLIC_VARIABLE = 'symbolic_variable'**

**CALL = 'call'**

**RETURN = 'return'**

**SIMPROCEDURE = 'simprocedure'**

**DIRTY = 'dirty'**

**SYSCALL = 'syscall'**

**CFG_HANDLE_JOB = 'cfg_handle_job'**

**VFG_HANDLE_SUCCESSOR = 'vfg_handle_successor'**

**ENGINE_PROCESS = 'engine_process'**

**MEMORY_PAGE_MAP = 'memory_page_map'**

__new__(*value*)

class angr.state_plugins.inspect.**InspectAttrs**

Bases: object

Per-event attributes published by the inspect machinery while a breakpoint is firing.

Each field is set on the active state before any matching breakpoint is checked, then cleared (back to None) after the event completes. Breakpoint actions read and write these fields through `state.inspect.attrs` to observe or override the in-flight event.

**vex_lift_addr:** Any = None

**vex_lift_size:** Any = None

**vex_lift_buff:** Any = None

**mem_read_address:** Any = None

**mem_read_expr:** Any = None

**mem_read_length:** Any = None

**mem_read_condition:** Any = None

**mem_read_endness:** Any = None

**mem_write_address:** Any = None

**mem_write_expr:** Any = None

**mem_write_length:** Any = None

**mem_write_condition:** Any = None

**mem_write_endness:** Any = None

**reg_read_offset:** Any = None

**reg_read_expr:** Any = None

**reg_read_length:** Any = None

**reg_read_condition:** Any = None

**reg_read_endness:** Any = None

**reg_write_offset:** Any = None

**reg_write_expr:** Any = None

**reg_write_length:** Any = None

**reg_write_condition:** Any = None

**reg_write_endness:** Any = None

**tmp_read_num:** Any = None

**tmp_read_expr:** Any = None

```
    tmp_write_num: Any = None
    tmp_write_expr: Any = None
    expr: Any = None
    expr_result: Any = None
    statement: Any = None
    instruction: Any = None
    address: Any = None
    added_constraints: Any = None
    function_address: Any = None
    exit_target: Any = None
    exit_guard: Any = None
    exit_jumpkind: Any = None
    backtrace: Any = None
    symbolic_name: Any = None
    symbolic_size: Any = None
    symbolic_expr: Any = None
    address_concretization_strategy: Any = None
    address_concretization_action: Any = None
    address_concretization_memory: Any = None
    address_concretization_expr: Any = None
    address_concretization_result: Any = None
    address_concretization_add_constraints: Any = None
    syscall_name: Any = None
    simprocedure_name: Any = None
    simprocedure_addr: Any = None
    simprocedure_result: Any = None
    simprocedure: Any = None
    dirty_name: Any = None
    dirty_handler: Any = None
    dirty_args: Any = None
    dirty_result: Any = None
```

**sim_engine:** *Any* = None

**sim_successors:** *Any* = None

**mapped_page:** *Any* = None

**mapped_address:** *Any* = None

**__init__**(*vex_lift_addr=None*, *vex_lift_size=None*, *vex_lift_buff=None*, *mem_read_address=None*, *mem_read_expr=None*, *mem_read_length=None*, *mem_read_condition=None*, *mem_read_endness=None*, *mem_write_address=None*, *mem_write_expr=None*, *mem_write_length=None*, *mem_write_condition=None*, *mem_write_endness=None*, *reg_read_offset=None*, *reg_read_expr=None*, *reg_read_length=None*, *reg_read_condition=None*, *reg_read_endness=None*, *reg_write_offset=None*, *reg_write_expr=None*, *reg_write_length=None*, *reg_write_condition=None*, *reg_write_endness=None*, *tmp_read_num=None*, *tmp_read_expr=None*, *tmp_write_num=None*, *tmp_write_expr=None*, *expr=None*, *expr_result=None*, *statement=None*, *instruction=None*, *address=None*, *added_constraints=None*, *function_address=None*, *exit_target=None*, *exit_guard=None*, *exit_jumpkind=None*, *backtrace=None*, *symbolic_name=None*, *symbolic_size=None*, *symbolic_expr=None*, *address_concretization_strategy=None*, *address_concretization_action=None*, *address_concretization_memory=None*, *address_concretization_expr=None*, *address_concretization_result=None*, *address_concretization_add_constraints=None*, *syscall_name=None*, *simprocedure_name=None*, *simprocedure_addr=None*, *simprocedure_result=None*, *simprocedure=None*, *dirty_name=None*, *dirty_handler=None*, *dirty_args=None*, *dirty_result=None*, *sim_engine=None*, *sim_successors=None*, *mapped_page=None*, *mapped_address=None*)

**Parameters**

- **vex_lift_addr** (*Any*)
- **vex_lift_size** (*Any*)
- **vex_lift_buff** (*Any*)
- **mem_read_address** (*Any*)
- **mem_read_expr** (*Any*)
- **mem_read_length** (*Any*)
- **mem_read_condition** (*Any*)
- **mem_read_endness** (*Any*)
- **mem_write_address** (*Any*)
- **mem_write_expr** (*Any*)
- **mem_write_length** (*Any*)
- **mem_write_condition** (*Any*)
- **mem_write_endness** (*Any*)
- **reg_read_offset** (*Any*)
- **reg_read_expr** (*Any*)
- **reg_read_length** (*Any*)
- **reg_read_condition** (*Any*)
- **reg_read_endness** (*Any*)

- **reg_write_offset** (*Any*)
- **reg_write_expr** (*Any*)
- **reg_write_length** (*Any*)
- **reg_write_condition** (*Any*)
- **reg_write_endness** (*Any*)
- **tmp_read_num** (*Any*)
- **tmp_read_expr** (*Any*)
- **tmp_write_num** (*Any*)
- **tmp_write_expr** (*Any*)
- **expr** (*Any*)
- **expr_result** (*Any*)
- **statement** (*Any*)
- **instruction** (*Any*)
- **address** (*Any*)
- **added_constraints** (*Any*)
- **function_address** (*Any*)
- **exit_target** (*Any*)
- **exit_guard** (*Any*)
- **exit_jumpkind** (*Any*)
- **backtrace** (*Any*)
- **symbolic_name** (*Any*)
- **symbolic_size** (*Any*)
- **symbolic_expr** (*Any*)
- **address_concretization_strategy** (*Any*)
- **address_concretization_action** (*Any*)
- **address_concretization_memory** (*Any*)
- **address_concretization_expr** (*Any*)
- **address_concretization_result** (*Any*)
- **address_concretization_add_constraints** (*Any*)
- **syscall_name** (*Any*)
- **simprocedure_name** (*Any*)
- **simprocedure_addr** (*Any*)
- **simprocedure_result** (*Any*)
- **simprocedure** (*Any*)
- **dirty_name** (*Any*)
- **dirty_handler** (*Any*)

- **dirty_args** (*Any*)
- **dirty_result** (*Any*)
- **sim_engine** (*Any*)
- **sim_successors** (*Any*)
- **mapped_page** (*Any*)
- **mapped_address** (*Any*)

**Return type**
None

class angr.state_plugins.inspect.**When**

Bases: StrEnum

When to trigger breakpoints.

**BEFORE = 'before'**

**AFTER = 'after'**

**BOTH = 'both'**

**__new__**(*value*)

type angr.state_plugins.inspect.**Action** = Callable[[*SimState*], None]

angr.state_plugins.inspect.**BP_IPDB**(*state*)

**Return type**
None

**Parameters**
**state** (SimState)

angr.state_plugins.inspect.**BP_IPYTHON**(*state*)

**Return type**
None

**Parameters**
**state** (SimState)

class angr.state_plugins.inspect.**BP**

Bases: object

A breakpoint.

**__init__**(*when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

**Parameters**

- **when** (When)
- **enabled** (*bool*)
- **condition** (*Callable[[SimState], bool] | None*)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

**check**(*state*, *when*)

Checks state *state* to see if the breakpoint should fire.

**Parameters**

- **state** (*SimState*) – The state.
- **when** (*When*) – Whether the check is happening before or after the event.

**Return type**
bool

**Returns**
A boolean representing whether the checkpoint should fire.

**fire**(*state*)

Trigger the breakpoint.

**Parameters**
**state** (*SimState*) – The state.

**class** angr.state_plugins.inspect.**SimInspector**

Bases: *SimStatePlugin*

The breakpoint interface, used to instrument execution. For usage information, look here: https://docs.angr.io/core-concepts/simulation#breakpoints

**BP_AFTER = 'after'**

**BP_BEFORE = 'before'**

**BP_BOTH = 'both'**

**__init__**()

**action**(*event_type*, *when*, ***kwargs*)

Called from within the engine when events happens. This function checks all breakpoints registered for that event and fires the ones whose conditions match.

**Return type**
None

**Parameters**

- **event_type** (EventType)
- **when** (When)
- **kwargs** (*dict[str, Any]*)

**make_breakpoint**(*event_type*, *when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

Creates and adds a breakpoint which would trigger on *event_type*. Additional arguments are passed to the *BP* constructor.

**Returns**
The created breakpoint, so that it can be removed later.

**Parameters**

- **event_type** (EventType)
- **when** (When)

- **enabled** (*bool*)
- **condition** (*Callable[[SimState], bool] | None*)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

**b**(*event_type*, *when=When.BEFORE*, *enabled=True*, *condition=None*, *action=<function BP_IPDB>*, ***kwargs*)

Creates and adds a breakpoint which would trigger on *event_type*. Additional arguments are passed to the *BP* constructor.

**Returns**
The created breakpoint, so that it can be removed later.

**Parameters**

- **event_type** (EventType)
- **when** (When)
- **enabled** (*bool*)
- **condition** (*Callable[[SimState], bool] | None*)
- **action** (Action)
- **kwargs** (*dict[str, Any]*)

**add_breakpoint**(*event_type*, *bp*)

Adds a breakpoint which would trigger on *event_type*.

**Parameters**

- **event_type** (*EventType*) – The event type to trigger on
- **bp** (*BP*) – The breakpoint

**Return type**
None

**Returns**
The created breakpoint.

**remove_breakpoint**(*event_type*, *bp=None*, *filter_func=None*)

Removes a breakpoint.

**Parameters**

- **bp** (*BP* | None) – The breakpoint to remove.
- **filter_func** (Callable[[*BP*], bool] | None) – A filter function to specify whether each breakpoint should be removed or not.
- **event_type** (EventType)

**Return type**
None

**downsize**()

Reset event-specific attributes on this plugin instance to save memory. This method is supposed to be called by breakpoint implementors. A typical workflow looks like the following:

```
>>> # Add `attr0` and `attr1` via the inspect machinery
>>> self.state.inspect(xxxxxx, attr0=yyyy, attr1=zzzz)
>>> # Get new attributes out of SimInspect in case they are modified by the user
>>> new_attr0 = self.state.inspect.attrs.attr0
>>> new_attr1 = self.state.inspect.attrs.attr1
>>> # Reset them
>>> self.state.inspect.downsize()
```

### 10.43.12 angr.state_plugins.javavm_classloader

**class** angr.state_plugins.javavm_classloader.**SimJavaVmClassloader**

Bases: *SimStatePlugin*

JavaVM Classloader is used as an interface for resolving and initializing Java classes.

**__init__**(*initialized_classes=None*)

**get_class**(*class_name*, *init_class=False*, *step_func=None*)

Get a class descriptor for the class.

**Parameters**

- **class_name** (*str*) – Name of class.
- **init_class** (*bool*) – Whether the class initializer <clinit> should be executed.
- **step_func** (*func*) – Callback function executed at every step of the simulation manager during the execution of the main <clinit> method

**get_superclass**(*class_*)

Get the superclass of the class.

**get_class_hierarchy**(*base_class*)

Walks up the class hierarchy and returns a list of all classes between base class (inclusive) and java.lang.Object (exclusive).

**is_class_initialized**(*class_*)

Indicates whether the classes initializing method <clinit> was already executed on the state.

**init_class**(*class_*, *step_func=None*)

This method simulates the loading of a class by the JVM, during which parts of the class (e.g. static fields) are initialized. For this, we run the class initializer method <clinit> (if available) and update the state accordingly.

**Note: Initialization is skipped, if the class has already been** initialized (or if it's not loaded in CLE).

**property initialized_classes**

List of all initialized classes.

### 10.43.13 angr.state_plugins.jni_references

**class** angr.state_plugins.jni_references.**SimStateJNIReferences**

Bases: *SimStatePlugin*

Management of the mapping between opaque JNI references and the corresponding Java objects.

**__init__**(*local_refs=None*, *global_refs=None*)

**lookup**(*opaque_ref*)

Lookups the object that was used for creating the reference.

**create_new_reference**(*obj*, *global_ref=False*)

Create a new reference thats maps to the given object.

**Parameters**

- **obj** – Object which gets referenced.
- **global_ref** (*bool*) – Whether a local or global reference is created.

**clear_local_references**()

Clear all local references.

**delete_reference**(*opaque_ref*, *global_ref=False*)

Delete the stored mapping of a reference.

**Parameters**

- **opaque_ref** – Reference which should be removed.
- **global_ref** (*bool*) – Whether opaque_ref is a local or global reference.

### 10.43.14 angr.state_plugins.libc

**class** angr.state_plugins.libc.**SimStateLibc**

Bases: *SimStatePlugin*

This state plugin keeps track of various libc stuff:

```
LOCALE_ARRAY = [b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x03
', b'\x02 ', b'\x02 ', b'\x02 ', b'\x02 ', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x02\x00',
b'\x02\x00', b'\x02\x00', b'\x02\x00', b'\x01`', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8',
b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x08\xd8', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5', b'\x08\xd5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5', b'\x08\xc5',
b'\x08\xc5', b'\x08\xc5', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x08\xd6', b'\x08\xd6', b'\x08\xd6', b'\x08\xd6',
b'\x08\xd6', b'\x08\xd6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6',
b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x08\xc6', b'\x04\xc0', b'\x04\xc0',
b'\x04\xc0', b'\x04\xc0', b'\x02\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00', b'\x00\x00',
```

TOLOWER_LOC_ARRAY = [128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 4294967295, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 97, 98,
99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115,
116, 117, 118, 119, 120, 121, 122, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101,
102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118,
119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152,
153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169,
170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186,
187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203,
204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220,
221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237,
238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254,
255]

TOUPPER_LOC_ARRAY = [128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156,
157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173,
174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190,
191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207,
208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241,
242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 4294967295, 0, 1,
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24,
25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66,
67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87,
88, 89, 90, 91, 92, 93, 94, 95, 96, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 123, 124, 125, 126, 127,
128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144,
145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161,
162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178,
179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195,
196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212,
213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229,
230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246,
247, 248, 249, 250, 251, 252, 253, 254, 255]

__init__()

property errno

ret_errno(*val*)

### 10.43.15 angr.state_plugins.light_registers

**class** angr.state_plugins.light_registers.**SimLightRegisters**

Bases: *SimStatePlugin*

**__init__**(*reg_map=None*, *registers=None*)

**resolve_register**(*offset*, *size*)

**load**(*offset*, *size=None*, ***kwargs*)

**store**(*offset*, *value*, *size=None*, *endness=None*, ***kwargs*)

### 10.43.16 angr.state_plugins.log

**class** angr.state_plugins.log.**SimStateLog**

Bases: *SimStatePlugin*

**__init__**(*log=None*)

**property actions**

**add_event**(*event_type*, ***kwargs*)

**add_action**(*action*)

**extend_actions**(*new_actions*)

**events_of_type**(*event_type*)

**actions_of_type**(*action_type*)

**property fresh_constraints**

**clear**()

### 10.43.17 angr.state_plugins.loop_data

**class** angr.state_plugins.loop_data.**SimStateLoopData**

Bases: *SimStatePlugin*

This class keeps track of loop-related information for states. Note that we have 2 counters for loop iterations (trip counts): the first recording the number of times one of the back edges (or continue edges) of a loop is taken, whereas the second recording the number of times the loop header (or loop entry) is executed. These 2 counters may differ since compilers usually optimize loops hence completely change the loop structure at the binary level. This is supposed to be used with *LoopSeer* exploration technique, which monitors loop execution. For the moment, the only thing we want to analyze is loop trip counts, but nothing prevents us from extending this plugin for other loop analyses.

**__init__**(*back_edge_trip_counts=None*, *header_trip_counts=None*, *current_loop=None*)

**Parameters**

- **back_edge_trip_counts** – Dictionary that stores back edge based trip counts for each loop. Keys are address of loop headers.
- **header_trip_counts** – Dictionary that stores header based trip counts for each loop. Keys are address of loop headers.
- **current_loop** – List of currently running loops. Each element is a tuple (loop object, list of loop exits).

### 10.43.18 angr.state_plugins.plugin

**class** angr.state_plugins.plugin.**SimStatePlugin**

Bases: object

This is a base class for SimState plugins. A SimState plugin will be copied along with the state when the state is branched. They are intended to be used for things such as tracking open files, tracking heap details, and providing storage and persistence for SimProcedures.

**__init__**()

**Return type**
None

**state:** *SimState*[Any, Any]

**set_state**(*state*)

Sets a new state (for example, if the state has been branched)

**Return type**
None

**static memo**(*f*)

A decorator function you should apply to copy

**Return type**
_CopyFunc[TypeVar(S_co)]

**Parameters**
**f** (*Callable[[Any, dict[int, Any]], S_co]*)

**copy**(*_memo*)

Should return a copy of the plugin without any state attached. Should check the memo first, and add itself to memo if it ends up making a new copy.

In order to simplify using the memo, you should annotate implementations of this function with SimStatePlugin.memo

The base implementation of this function constructs a new instance of the plugin's class without calling its initializer. If you super-call down to it, make sure you instantiate all the fields in your copy method!

**Parameters**

- **memo** – A dictionary mapping object identifiers (id(obj)) to their copied instance. Use this to avoid infinite recursion and diverged copies.
- **_memo** (*dict[int, Any]*)

**Return type**
*SimStatePlugin*

**merge**(*others*, *merge_conditions*, *common_ancestor=None*)

Should merge the state plugin with the provided others. This will be called by state.merge() after copying the target state, so this should mutate the current instance to merge with the others.

Note that when multiple instances of a single plugin object (for example, a file) are referenced in the state, it is important that merge only ever be called once. This should be solved by designating one of the plugin's referees as the "real owner", who should be the one to actually merge it. This technique doesn't work to resolve the similar issue that arises during copying because merging doesn't produce a new reference to insert.

There will be n `others` and n+1 merge conditions, since the first condition corresponds to self. To match elements up to conditions, say `zip([self] + others, merge_conditions)`

When implementing this, make sure that you "deepen" both `others` and `common_ancestor` before calling sub-elements' merge methods, e.g.

```
self.foo.merge(
    [o.foo for o in others],
    merge_conditions,
    common_ancestor=common_ancestor.foo if common_ancestor is not None else None
)
```

During static analysis, merge_conditions can be None, in which case you should use `state.solver.union(values)`. TODO: fish please make this less bullshit

There is a utility `claripy.ite_cases` which will help with constructing arbitrarily large merged ASTs. Use it like `self.bar = claripy.ite_cases(zip(conditions[1:], [o.bar for o in others]), self.bar)`

**Parameters**

- **others** – the other state plugins to merge with
- **merge_conditions** – a symbolic condition for each of the plugins
- **common_ancestor** – a common ancestor of this plugin and the others being merged

**Returns**
True if the state plugins are actually merged.

**Return type**
bool

**classmethod register_default**(*name*, *xtr=None*)

**Return type**
None

**Parameters**

- **name** (*str*)
- **xtr** (*type[SimStatePlugin] | str | None*)

**init_state**()

Use this function to perform any initialization on the state at plugin-add time

**Return type**
None

### 10.43.19 angr.state_plugins.posix

**class** angr.state_plugins.posix.**PosixDevFS**

Bases: *SimMount*

**class** angr.state_plugins.posix.**PosixProcFS**

Bases: *SimMount*

The virtual file system mounted at /proc (as of now, on Linux).

class angr.state_plugins.posix.**SimSystemPosix**

Bases: *SimStatePlugin*

Data storage and interaction mechanisms for states with an environment conforming to posix. Available as `state.posix`.

**SIG_BLOCK = 0**

**SIG_UNBLOCK = 1**

**SIG_SETMASK = 2**

**EPERM = 1**

**ENOENT = 2**

**ESRCH = 3**

**EINTR = 4**

**EIO = 5**

**ENXIO = 6**

**E2BIG = 7**

**ENOEXEC = 8**

**EBADF = 9**

**ECHILD = 10**

**EAGAIN = 11**

**ENOMEM = 12**

**EACCES = 13**

**EFAULT = 14**

**ENOTBLK = 15**

**EBUSY = 16**

**EEXIST = 17**

**EXDEV = 18**

**ENODEV = 19**

**ENOTDIR = 20**

**EISDIR = 21**

**EINVAL = 22**

**ENFILE = 23**

**EMFILE = 24**

**ENOTTY = 25**

**ETXTBSY = 26**

**EFBIG = 27**

**ENOSPC = 28**

**ESPIPE = 29**

**EROFS = 30**

**EMLINK = 31**

**EPIPE = 32**

**EDOM = 33**

**ERANGE = 34**

**__init__**(*stdin=None*, *stdout=None*, *stderr=None*, *fd=None*, *sockets=None*, *socket_queue=None*, *argv=None*, *argc=None*, *environ=None*, *auxv=None*, *tls_modules=None*, *sigmask=None*, *pid=None*, *ppid=None*, *uid=None*, *gid=None*, *brk=None*)

**property closed_fds**

**set_brk**(*new_brk*)

**open**(*name*, *flags*, *preferred_fd=None*)

Open a symbolic file. Basically open(2).

**Parameters**

- **name** (*string or bytes*) – Path of the symbolic file, as a string or bytes.
- **flags** – File operation flags, a bitfield of constants from open(2), as an AST
- **preferred_fd** – Assign this fd if it's not already claimed.

**Returns**

The file descriptor number allocated (maps through posix.get_fd to a SimFileDescriptor) or -1 if the open fails.

`mode` from open(2) is unsupported at present.

**open_socket**(*ident*)

**get_fd**(*fd*, *create_file=True*)

Looks up the SimFileDescriptor associated with the given number (an AST). If the number is concrete and does not map to anything, return None. If the number is symbolic, constrain it to an open fd and create a new file for it. Set create_file to False if no write-access is planned (i.e. fd is read-only).

**get_concrete_fd**(*fd*, *create_file=True*)

Same behavior as get_fd(fd), only the result is a concrete integer fd (or -1) instead of a SimFileDescriptor.

**close**(*fd*)

Closes the given file descriptor (an AST). Returns whether the operation succeeded (a concrete boolean)

**fstat**(*fd*)

**fstat_with_result**(*sim_fd*)

**sigmask**(*sigsetsize=None*)

Gets the current sigmask. If it's blank, a new one is created (of sigsetsize).

**Parameters**
**sigsetsize** – the size (in *bytes* of the sigmask set)

**Returns**
the sigmask

**sigprocmask**(*how*, *new_mask*, *sigsetsize*, *valid_ptr=True*)

Updates the signal mask.

**Parameters**

- **how** – the "how" argument of sigprocmask (see manpage)
- **new_mask** – the mask modification to apply
- **sigsetsize** – the size (in *bytes* of the sigmask set)
- **valid_ptr** – is set if the new_mask was not NULL

**dump_file_by_path**(*path*, ***kwargs*)

Returns the concrete content for a file by path.

**Parameters**

- **path** – file path as string
- **kwargs** – passed to state.solver.eval

**Returns**
file contents as string

**dumps**(*fd*, ***kwargs*)

Returns the concrete content for a file descriptor.

BACKWARD COMPATIBILITY: if you ask for file descriptors 0 1 or 2, it will return the data from stdin, stdout, or stderr as a flat string.

**Parameters**
**fd** – A file descriptor.

**Returns**
The concrete content.

**Return type**
str

### 10.43.20 angr.state_plugins.preconstrainer

**class** angr.state_plugins.preconstrainer.**SimStatePreconstrainer**

Bases: *SimStatePlugin*

This state plugin manages the concept of preconstraining - adding constraints which you would like to remove later.

**Parameters**
**constrained_addrs** – SimActions for memory operations whose addresses should be constrained during crash analysis

**__init__**(*constrained_addrs=None*)

**preconstrain**(*value*, *variable*)

Add a preconstraint that `variable == value` to the state.

**Parameters**

- **value** – The concrete value. Can be a bitvector or a bytestring or an integer.
- **variable** – The BVS to preconstrain.

**preconstrain_file**(*content*, *simfile*, *set_length=False*)

Preconstrain the contents of a file.

**Parameters**

- **content** – The content to preconstrain the file to. Can be a bytestring or a list thereof.
- **simfile** – The actual simfile to preconstrain

**preconstrain_flag_page**(*magic_content*)

Preconstrain the data in the flag page.

**Parameters**
**magic_content** – The content of the magic page as a bytestring.

**remove_preconstraints**(*to_composite_solver=True*, *simplify=True*)

Remove the preconstraints from the state.

If you are using the zen plugin, this will also use that to filter the constraints.

**Parameters**

- **to_composite_solver** – Whether to convert the replacement solver to a composite solver. You probably want this if you're switching from tracing to symbolic analysis.
- **simplify** – Whether to simplify the resulting set of constraints.

**reconstrain**()

Split the solver. If any of the subsolvers time out after a short timeout (10 seconds), re-add the preconstraints associated with each of its variables. Hopefully these constraints still allow us to do meaningful things to the state.

### 10.43.21 angr.state_plugins.scratch

**class** angr.state_plugins.scratch.**SimStateScratch**

Bases: *SimStatePlugin*

Implements the scratch state plugin.

**__init__**(*scratch=None*)

**property ail_block_idx**

**property is_ail**

**property priv**

**push_priv**(*priv*)

**pop_priv**()

**set_tyenv**(*tyenv*)

`tmp_expr`(*tmp*)

Returns the Claripy expression of a VEX temp value.

**Parameters**

- **tmp** – the number of the tmp
- **simplify** – simplify the tmp before returning it

**Returns**
a Claripy expression of the tmp

`store_tmp`(*tmp*, *content*, *reg_deps=frozenset({})*, *tmp_deps=frozenset({})*, *deps=None*, ***kwargs*)

Stores a Claripy expression in a VEX temp value. If in symbolic mode, this involves adding a constraint for the tmp's symbolic variable.

**Parameters**

- **tmp** – the number of the tmp
- **content** – a Claripy expression of the content
- **reg_deps** – the register dependencies of the content
- **tmp_deps** – the temporary value dependencies of the content

`clear`()

### 10.43.22 angr.state_plugins.sim_action

**class** angr.state_plugins.sim_action.**SimAction**

Bases: *SimEvent*

A SimAction represents a semantic action that an analyzed program performs.

`TMP = 'tmp'`

`REG = 'reg'`

`MEM = 'mem'`

`__init__`(*state*, *region_type*)

Initializes the SimAction.

**Parameters**
**state** – the state that's the SimAction is taking place in.

`property all_objects`

`property is_symbolic`

`property tmp_deps`

`property reg_deps`

`copy`()

`downsize`()

Clears some low-level details (that take up memory) out of the SimAction.

class angr.state_plugins.sim_action.**SimActionExit**

Bases: *SimAction*

An Exit action represents a (possibly conditional) jump.

**CONDITIONAL = 'conditional'**

**DEFAULT = 'default'**

**property all_objects**

**property is_symbolic**

class angr.state_plugins.sim_action.**SimActionConstraint**

Bases: *SimAction*

A constraint action represents an extra constraint added during execution of a path.

**property all_objects**

**property is_symbolic**

class angr.state_plugins.sim_action.**SimActionOperation**

Bases: *SimAction*

An action representing an operation between variables and/or constants.

**property all_objects**

**property is_symbolic**

class angr.state_plugins.sim_action.**SimActionData**

Bases: *SimAction*

A Data action represents a read or a write from memory, registers or a file.

**READ = 'read'**

**WRITE = 'write'**

**OPERATE = 'operate'**

**property all_objects**

**property is_symbolic**

**property tmp_deps**

**property reg_deps**

**property storage**

### 10.43.23 angr.state_plugins.sim_action_object

angr.state_plugins.sim_action_object.**ast_preserving_op**(*f*, **args*)

angr.state_plugins.sim_action_object.**ast_stripping_decorator**(*f*)

class angr.state_plugins.sim_action_object.**SimActionObject**

Bases: object

A SimActionObject tracks an AST and its dependencies.

**__init__**(*ast*, *reg_deps=frozenset({})*, *tmp_deps=frozenset({})*, *deps=frozenset({})*, *state=None*)

**Parameters**

- **ast** (*Base*)
- **reg_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **tmp_deps** (*frozenset[SimActionData | SimActionOperation]*)
- **deps** (*frozenset*)
- **state** (*SimState | None*)

**ast: Base**

**reg_deps: frozenset[*SimActionData* | *SimActionOperation*]**

**tmp_deps: frozenset[*SimActionData* | *SimActionOperation*]**

**to_claripy**()

**Return type**
Base

**copy**()

**Return type**
*SimActionObject*

**is_leaf**()

**Return type**
bool

**property op: str**

**property args: tuple[ArgType, ...]**

**property length: int | None**

**property variables: frozenset[str]**

**property symbolic: bool**

**property annotations: tuple[Annotation, ...]**

**property depth: int**

**SDiv**(*other*)

**Return type**
*SimActionObject*

**SMod**(*other*)

**Return type**
*SimActionObject*

**union**(*other*)

**Return type**
*SimActionObject*

**intersection**(*other*)

**Return type**
*SimActionObject*

**widen**(*other*)

**Return type**
*SimActionObject*

**raw_to_bv**()

**Return type**
*SimActionObject*

**bv_to_fp**()

**Return type**
*SimActionObject*

### 10.43.24 angr.state_plugins.sim_event

**class** angr.state_plugins.sim_event.**SimEvent**

Bases: object

A SimEvent is a log entry for some notable event during symbolic execution. It logs the location it was generated (ins_addr, bbl_addr, stmt_idx, and sim_procedure) as well as arbitrary tags (objects).

You may also be interested in SimAction, which is a specialization of SimEvent for CPU events.

**__init__**(*state*, *event_type*, ***kwargs*)

angr.state_plugins.sim_event.**resource_event**(*state*, *exception*)

### 10.43.25 angr.state_plugins.solver

angr.state_plugins.solver.**timed_function**(*f*)

angr.state_plugins.solver.**enable_timing**()

angr.state_plugins.solver.**disable_timing**()

angr.state_plugins.solver.**error_converter**(*f*)

angr.state_plugins.solver.**concrete_path_bool**(*f*)

angr.state_plugins.solver.**concrete_path_not_bool**(*f*)

angr.state_plugins.solver.**concrete_path_scalar**(*f*)

angr.state_plugins.solver.**concrete_path_tuple**(*f*)

angr.state_plugins.solver.**concrete_path_list**(*f*)

**class** angr.state_plugins.solver.**SimSolver**

Bases: *SimStatePlugin*

This is the plugin you'll use to interact with symbolic variables, creating them and evaluating them. It should be available on a state as `state.solver`.

Any top-level variable of the claripy module can be accessed as a property of this object.

**__init__**(*solver=None*, *all_variables=None*, *temporal_tracked_variables=None*, *eternal_tracked_variables=None*)

**reload_solver**(*constraints=None*)

Reloads the solver. Useful when changing solver options.

**Parameters**

**constraints** (*list*) – A new list of constraints to use in the reloaded solver instead of the current one

**get_variables**(**keys*)

Iterate over all variables for which their tracking key is a prefix of the values provided.

Elements are a tuple, the first element is the full tracking key, the second is the symbol.

```
>>> list(s.solver.get_variables('mem'))
[(('mem', 0x1000), <BV64 mem_1000_4_64>), (('mem', 0x1008), <BV64 mem_1008_5_64>
↪)]
```

```
>>> list(s.solver.get_variables('file'))
[(('file', 1, 0), <BV8 file_1_0_6_8>), (('file', 1, 1), <BV8 file_1_1_7_8>),
    (('file', 2, 0), <BV8 file_2_0_8_8>)]
```

```
>>> list(s.solver.get_variables('file', 2))
[(('file', 2, 0), <BV8 file_2_0_8_8>)]
```

```
>>> list(s.solver.get_variables())
[(('mem', 0x1000), <BV64 mem_1000_4_64>), (('mem', 0x1008), <BV64 mem_1008_5_64>
↪),
    (('file', 1, 0), <BV8 file_1_0_6_8>), (('file', 1, 1), <BV8 file_1_1_7_8>),
    (('file', 2, 0), <BV8 file_2_0_8_8>)]
```

**register_variable**(*v*, *key*, *eternal=True*)

Register a value with the variable tracking system

**Parameters**

- **v** – The BVS to register
- **key** – A tuple to register the variable under

**Parma eternal**

Whether this is an eternal variable, default True. If False, an incrementing counter will be appended to the key.

**describe_variables**(*v*)

Given an AST, iterate over all the keys of all the BVS leaves in the tree which are registered.

**Unconstrained**(*name*, *bits*, *uninitialized=True*, *inspect=True*, *events=True*, *key=None*, *eternal=False*, *uc_alloc_depth=None*, ***kwargs*)

Creates an unconstrained symbol or a default concrete value (0), based on the state options.

**Parameters**

- **name** – The name of the symbol.
- **bits** – The size (in bits) of the symbol.
- **uninitialized** – Whether this value should be counted as an "uninitialized" value in the course of an analysis.
- **inspect** – Set to False to avoid firing SimInspect breakpoints
- **events** – Set to False to avoid generating a SimEvent for the occasion
- **key** – Set this to a tuple of increasingly specific identifiers (for example, `('mem', 0xffbeff00)` or `('file', 4, 0x20)` to cause it to be tracked, i.e. accessible through `solver.get_variables`.
- **eternal** – Set to True in conjunction with setting a key to cause all states with the same ancestry to retrieve the same symbol when trying to create the value. If False, a counter will be appended to the key.

**Returns**

an unconstrained symbol (or a concrete value of 0).

**BVS**(*name*, *size*, *min=None*, *max=None*, *stride=None*, *uninitialized=False*, *explicit_name=False*, *key=None*, *eternal=False*, *inspect=True*, *events=True*, ***kwargs*)

Creates a bit-vector symbol (i.e., a variable). Other keyword parameters are passed directly on to the constructor of claripy.ast.BV.

**Parameters**

- **name** – The name of the symbol.
- **size** – The size (in bits) of the bit-vector.
- **min** – The minimum value of the symbol. Note that this **only** work when using VSA.
- **max** – The maximum value of the symbol. Note that this **only** work when using VSA.
- **stride** – The stride of the symbol. Note that this **only** work when using VSA.
- **uninitialized** – Whether this value should be counted as an "uninitialized" value in the course of an analysis.
- **explicit_name** – Set to True to prevent an identifier from appended to the name to ensure uniqueness.
- **key** – Set this to a tuple of increasingly specific identifiers (for example, `('mem', 0xffbeff00)` or `('file', 4, 0x20)` to cause it to be tracked, i.e. accessible through `solver.get_variables`.
- **eternal** – Set to True in conjunction with setting a key to cause all states with the same ancestry to retrieve the same symbol when trying to create the value. If False, a counter will be appended to the key.
- **inspect** – Set to False to avoid firing SimInspect breakpoints
- **events** – Set to False to avoid generating a SimEvent for the occasion

**Returns**

A BV object representing this symbol.

**downsize**()

Frees memory associated with the constraint solver by clearing all of its internal caches.

**property constraints**

Returns the constraints of the state stored by the solver.

**eval_to_ast**(*e*, *n*, *extra_constraints=()*, *exact=None*)

Evaluate an expression, using the solver if necessary. Returns AST objects.

**Parameters**

- **e** – the expression
- **n** – the number of desired solutions
- **extra_constraints** – extra constraints to apply to the solver
- **exact** – if False, returns approximate solutions

**Returns**
a tuple of the solutions, in the form of claripy AST nodes

**Return type**
tuple

**max**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the maximum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the maximum possible value of e (backend object)

**min**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the minimum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the minimum possible value of e (backend object)

**solution**(*e*, *v*, *extra_constraints=()*, *exact=None*)

Return True if *v* is a solution of *expr* with the extra constraints, False otherwise.

**Parameters**

- **e** – An expression (an AST) to evaluate
- **v** – The proposed solution (an AST)
- **extra_constraints** – Extra constraints (as ASTs) to add to the solver for this solve.
- **exact** – If False, return approximate solutions.

**Returns**
True if *v* is a solution of *expr*, False otherwise

**is_true**(*e*, *extra_constraints=()*, *exact=None*)

If the expression provided is absolutely, definitely a true boolean, return True. Note that returning False doesn't necessarily mean that the expression can be false, just that we couldn't figure that out easily.

**Parameters**

- **e** – An expression (an AST) to evaluate

- **extra_constraints** – Extra constraints (as ASTs) to add to the solver for this solve.
- **exact** – If False, return approximate solutions.

**Returns**
True if *v* is definitely true, False otherwise

**is_false**(*e*, *extra_constraints=()*, *exact=None*)

If the expression provided is absolutely, definitely a false boolean, return True. Note that returning False doesn't necessarily mean that the expression can be true, just that we couldn't figure that out easily.

**Parameters**

- **e** – An expression (an AST) to evaluate
- **extra_constraints** – Extra constraints (as ASTs) to add to the solver for this solve.
- **exact** – If False, return approximate solutions.

**Returns**
True if *v* is definitely false, False otherwise

**unsat_core**(*extra_constraints=()*)

This function returns the unsat core from the backend solver.

**Parameters**
**extra_constraints** – Extra constraints (as ASTs) to add to the solver for this solve.

**Returns**
The unsat core.

**satisfiable**(*extra_constraints=()*, *exact=None*)

This function does a constraint check and checks if the solver is in a sat state.

**Parameters**

- **extra_constraints** – Extra constraints (as ASTs) to add to s for this solve
- **exact** – If False, return approximate solutions.

**Returns**
True if sat, otherwise false

**add**(**constraints*)

Add some constraints to the solver.

**Parameters**
**constraints** – Pass any constraints that you want to add (ASTs) as varargs.

**CastType = ~CastType**

**eval_upto**(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression, using the solver if necessary. Returns primitives as specified by the *cast_to* parameter. Only certain primitives are supported, check the implementation of *_cast_to* to see which ones.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]

- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- **e** – the expression
- **n** – the number of desired solutions
- **extra_constraints** – extra constraints to apply to the solver
- **exact** – if False, returns approximate solutions
- **cast_to** – desired type of resulting values

**Returns**
a tuple of the solutions, in the form of Python primitives

**Return type**
tuple

**eval**(*e*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get any possible solution. The desired output types can be specified using the *cast_to* parameter. *extra_constraints* can be used to specify additional constraints the returned values must satisfy.

**Overloads**

- **self**, **e** (claripy.ast.BV), **cast_to** (None), **kwargs** → int
- **self**, **e** (claripy.ast.BV), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.Bool), **cast_to** (None), **kwargs** → bool
- **self**, **e** (claripy.ast.Bool), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.FP), **cast_to** (None), **kwargs** → float
- **self**, **e** (claripy.ast.FP), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.String | str), **cast_to** (None), **kwargs** → str
- **self**, **e** (_EvalArg), **cast_to** (None), **kwargs** → Any
- **self**, **e** (_EvalArg), **cast_to** (type[CastType]), **kwargs** → CastType

**Parameters**

- **e** – the expression to get a solution for
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*
- **cast_to** – desired type of resulting values

**Raises**
***SimUnsatError*** – if no solution could be found satisfying the given constraints

**Returns**

**eval_one**(*e*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get the only possible solution. Errors if either no or more than one solution is returned. A kwarg parameter *default* can be specified to be returned instead of failure!

**Overloads**

- **self**, **e** (claripy.ast.BV), **cast_to** (None), **kwargs** → int
- **self**, **e** (claripy.ast.BV), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.Bool), **cast_to** (None), **kwargs** → bool
- **self**, **e** (claripy.ast.Bool), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.FP), **cast_to** (None), **kwargs** → float
- **self**, **e** (claripy.ast.FP), **cast_to** (type[CastType]), **kwargs** → CastType
- **self**, **e** (claripy.ast.String | str), **cast_to** (None), **kwargs** → str
- **self**, **e** (_EvalArg), **cast_to** (None), **kwargs** → Any
- **self**, **e** (_EvalArg), **cast_to** (type[CastType]), **kwargs** → CastType

**Parameters**

- **e** – the expression to get a solution for
- **cast_to** – desired type of resulting values
- **default** – A value can be passed as a kwarg here. It will be returned in case of failure.
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if more than one solution was found to satisfy the given constraints

**Returns**

The value for *e*

**eval_atmost**(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get at most *n* possible solutions. Errors if either none or more than *n* solutions are returned.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- **e** – the expression to get a solution for
- **n** – the inclusive upper limit on the number of solutions
- **cast_to** – desired type of resulting values

- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if more than *n* solutions were found to satisfy the given constraints

**Returns**
The solutions for *e*

**eval_atleast**(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get at least *n* possible solutions. Errors if less than *n* solutions were found.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- **e** – the expression to get a solution for
- **n** – the inclusive lower limit on the number of solutions
- **cast_to** – desired type of resulting values
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if less than *n* solutions were found to satisfy the given constraints

**Returns**
The solutions for *e*

**eval_exact**(*e*, *n*, *cast_to=None*, ***kwargs*)

Evaluate an expression to get exactly the *n* possible solutions. Errors if any number of solutions other than *n* was found to exist.

**Overloads**

- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (None), **kwargs** → list[int]
- **self**, **e** (claripy.ast.BV), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (None), **kwargs** → list[bool]
- **self**, **e** (claripy.ast.Bool), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (None), **kwargs** → list[float]

- **self**, **e** (claripy.ast.FP), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]
- **self**, **e** (claripy.ast.String | str), **n** (int), **cast_to** (None), **kwargs** → list[str]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (None), **kwargs** → list[Any]
- **self**, **e** (_EvalArg), **n** (int), **cast_to** (type[CastType]), **kwargs** → list[CastType]

**Parameters**

- **e** – the expression to get a solution for
- **n** – the inclusive lower limit on the number of solutions
- **cast_to** – desired type of resulting values
- **kwargs** – Any additional kwargs will be passed down to *eval_upto*

**Raises**

- ***SimUnsatError*** – if no solution could be found satisfying the given constraints
- ***SimValueError*** – if any number of solutions other than *n* were found to satisfy the given constraints

**Returns**

The solutions for *e*

**min_int**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the minimum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the minimum possible value of e (backend object)

**max_int**(*e*, *extra_constraints=()*, *exact=None*, *signed=False*)

Return the maximum value of expression *e*.

:param e : expression (an AST) to evaluate :type extra_constraints: :param extra_constraints: extra constraints (as ASTs) to add to the solver for this solve :param exact : if False, return approximate solutions. :param signed : Whether the expression should be treated as a signed value. :return: the maximum possible value of e (backend object)

**unique**(*e*, ***kwargs*)

Returns True if the expression *e* has only one solution by querying the constraint solver. It does also add that unique solution to the solver's constraints.

**symbolic**(*e*)

Returns True if the expression *e* is symbolic.

**single_valued**(*e*)

Returns True whether *e* is a concrete value or is a value set with only 1 possible value. This differs from *unique* in that this *does* not query the constraint solver.

**simplify**(*e=None*)

Simplifies *e*. If *e* is None, simplifies the constraints of this state.

**variables**(*e*)

Returns the symbolic variables present in the AST of *e*.

### 10.43.26 angr.state_plugins.symbolizer

**class** angr.state_plugins.symbolizer.**SimSymbolizer**

Bases: *SimStatePlugin*

The symbolizer state plugin ensures that pointers that are stored in memory are symbolic. This allows for the tracking of and reasoning over these pointers (for example, to reason about memory disclosure).

**__init__**()

**set_symbolization_for_all_pages**()

Sets the symbolizer to symbolize pointers to all pages as they are written to memory..

**set_symbolized_target_range**(*base*, *length*)

All pointers to the target range will be symbolized as they are written to memory.

Due to optimizations, the _pages_ containing this range will be set as symbolization targets, not just the range itself.

**resymbolize**()

Re-symbolizes all pointers in memory. This can be called to symbolize any pointers to target regions that were written (and not mangled beyond recognition) before symbolization was set.

### 10.43.27 angr.state_plugins.trace_additions

This file contains objects to track additional information during a trace or modify symbolic variables during a trace.

The ChallRespInfo plugin tracks variables in stdin and stdout to enable handling of challenge response It handles atoi/int2str in a special manner since path constraints will usually prevent their values from being modified

The Zen plugin simplifies expressions created from variables in the flag page (losing some accuracy) to avoid situations where they become to complex for z3, but the actual equation doesn't matter much. This can happen in challenge response if all of the values in the flag page are multiplied together before being printed.

**class** angr.state_plugins.trace_additions.**FormatInfo**

Bases: object

**copy**()

**compute**(*state*)

**get_type**()

**class** angr.state_plugins.trace_additions.**FormatInfoStrToInt**

Bases: *FormatInfo*

**__init__**(*addr*, *func_name*, *str_arg_num*, *base*, *base_arg*, *allows_negative*)

**copy**()

**compute**(*state*)

**get_type**()

**class** angr.state_plugins.trace_additions.**FormatInfoIntToStr**

Bases: *FormatInfo*

**__init__**(*addr*, *func_name*, *int_arg_num*, *str_dst_num*, *base*, *base_arg*)

**copy**()

**compute**(*state*)

**get_type**()

class angr.state_plugins.trace_additions.**FormatInfoDontConstrain**

Bases: *FormatInfo*

**__init__**(*addr*, *func_name*, *check_symbolic_arg*)

**copy**()

**compute**(*state*)

**get_type**()

angr.state_plugins.trace_additions.**int2base**(*x*, *base*)

angr.state_plugins.trace_additions.**generic_info_hook**(*state*)

angr.state_plugins.trace_additions.**end_info_hook**(*state*)

angr.state_plugins.trace_additions.**exit_hook**(*state*)

angr.state_plugins.trace_additions.**syscall_hook**(*state*)

angr.state_plugins.trace_additions.**constraint_hook**(*state*)

class angr.state_plugins.trace_additions.**ChallRespInfo**

Bases: *SimStatePlugin*

This state plugin keeps track of the reads and writes to symbolic addresses

**__init__**()

**static get_byte**(*var_name*)

**lookup_original**(*replacement*)

**pop_from_backup**()

**get_stdin_indices**(*variable*)

**get_stdout_indices**(*variable*)

**get_real_len**(*input_val*, *base*, *result_bv*, *allows_negative*)

**get_possible_len**(*input_val*, *base*, *allows_negative*)

**get_same_length_constraints**()

**static atoi_dumps**(*state*, *require_same_length=True*)

**static prep_tracer**(*state*, *format_infos=None*)

angr.state_plugins.trace_additions.**zen_hook**(*state*, *expr*)

angr.state_plugins.trace_additions.**zen_memory_write**(*state*)

angr.state_plugins.trace_additions.**zen_register_write**(*state*)

class angr.state_plugins.trace_additions.**ZenPlugin**

Bases: *SimStatePlugin*

**__init__**(*max_depth=13*)

**static get_flag_rand_args**(*expr*)

**get_expr_depth**(*expr*)

**get_flag_bytes**(*ast*)

**filter_constraints**(*constraints*)

**analyze_transmit**(*state*, *buf*)

**static prep_tracer**(*state*)

### 10.43.28 angr.state_plugins.uc_manager

class angr.state_plugins.uc_manager.**SimUCManager**

Bases: *SimStatePlugin*

**__init__**(*man=None*)

**assign**(*dst_addr_ast*)

Assign a new region for under-constrained symbolic execution.

**Parameters**
**dst_addr_ast** – the symbolic AST which address of the new allocated region will be assigned to.

**Returns**
as ast of memory address that points to a new region

**get_alloc_depth**(*addr*)

**Return type**
int | None

**Parameters**
**addr** (*int | Base*)

**set_alloc_depth**(*addr*, *depth*)

**Parameters**

- **addr** (*Base*)
- **depth** (*int*)

**is_bounded**(*ast*)

Test whether an AST is bounded by any existing constraint in the related solver.

**Parameters**
**ast** – an claripy.AST object

**Returns**
True if there is at least one related constraint, False otherwise

### 10.43.29 angr.state_plugins.unicorn_engine

class angr.state_plugins.unicorn_engine.**MEM_PATCH**

Bases: Structure

struct mem_update_t

**address**

Structure/Union member

**length**

Structure/Union member

**next**

Structure/Union member

class angr.state_plugins.unicorn_engine.**TRANSMIT_RECORD**

Bases: Structure

struct transmit_record_t

**count**

Structure/Union member

**data**

Structure/Union member

**fd**

Structure/Union member

class angr.state_plugins.unicorn_engine.**TaintEntityEnum**

Bases: object

taint_entity_enum_t

**TAINT_ENTITY_REG = 0**

**TAINT_ENTITY_TMP = 1**

**TAINT_ENTITY_MEM = 2**

**TAINT_ENTITY_NONE = 3**

class angr.state_plugins.unicorn_engine.**MemoryValue**

Bases: Structure

struct memory_value_t

**address**

Structure/Union member

**is_value_set**

Structure/Union member

**is_value_symbolic**

Structure/Union member

**value**

Structure/Union member

class angr.state_plugins.unicorn_engine.**RegisterValue**

Bases: Structure

struct register_value_t

**offset**

Structure/Union member

**size**

Structure/Union member

**value**

Structure/Union member

class angr.state_plugins.unicorn_engine.**VEXStmtDetails**

Bases: Structure

struct sym_vex_stmt_details_t

**has_memory_dep**

Structure/Union member

**memory_values**

Structure/Union member

**memory_values_count**

Structure/Union member

**stmt_idx**

Structure/Union member

class angr.state_plugins.unicorn_engine.**BlockDetails**

Bases: Structure

struct sym_block_details_ret_t

**block_addr**

Structure/Union member

**block_size**

Structure/Union member

**block_trace_ind**

Structure/Union member

**has_symbolic_exit**

Structure/Union member

**register_values**

Structure/Union member

**register_values_count**

Structure/Union member

**symbolic_vex_stmts**

Structure/Union member

**symbolic_vex_stmts_count**

Structure/Union member

class angr.state_plugins.unicorn_engine.STOP

Bases: object

enum stop_t

STOP_NORMAL = 0

STOP_STOPPOINT = 1

STOP_ERROR = 2

STOP_SYSCALL = 3

STOP_EXECNONE = 4

STOP_ZEROPAGE = 5

STOP_NOSTART = 6

STOP_SEGFAULT = 7

STOP_ZERO_DIV = 8

STOP_NODECODE = 9

STOP_HLT = 10

STOP_VEX_LIFT_FAILED = 11

STOP_SYMBOLIC_PC = 12

STOP_SYMBOLIC_READ_ADDR = 13

STOP_SYMBOLIC_READ_SYMBOLIC_TRACKING_DISABLED = 14

STOP_SYMBOLIC_WRITE_ADDR = 15

STOP_SYMBOLIC_BLOCK_EXIT_CONDITION = 16

STOP_SYMBOLIC_BLOCK_EXIT_TARGET = 17

STOP_UNSUPPORTED_STMT_PUTI = 18

STOP_UNSUPPORTED_STMT_STOREG = 19

STOP_UNSUPPORTED_STMT_LOADG = 20

STOP_UNSUPPORTED_STMT_CAS = 21

STOP_UNSUPPORTED_STMT_LLSC = 22

STOP_UNSUPPORTED_STMT_DIRTY = 23

STOP_UNSUPPORTED_EXPR_GETI = 24

STOP_UNSUPPORTED_STMT_UNKNOWN = 25

STOP_UNSUPPORTED_EXPR_UNKNOWN = 26

STOP_UNKNOWN_MEMORY_WRITE_SIZE = 27

STOP_SYSCALL_ARM = 28

STOP_X86_CPUID = 29

stop_message = {0: 'Reached maximum steps', 1: 'Hit a stop point', 2: 'Something wrong', 3: 'Unable to handle syscall', 4: 'Fetching empty page', 5: 'Accessing zero page', 6: 'Failed to start', 7: 'Permissions or mapping error', 8: 'Divide by zero', 9: 'Instruction decoding error', 10: 'hlt instruction encountered', 11: 'Failed to lift block to VEX', 12: 'Instruction pointer became symbolic', 13: 'Attempted to read from symbolic address', 14: 'Attempted to read symbolic data from memory but symbolic tracking is disabled', 15: 'Attempted to write to symbolic address', 16: "Guard condition of block's exit statement is symbolic", 17: 'Target of default exit of block is symbolic', 18: 'Symbolic taint propagation for PutI statement not yet supported', 19: 'Symbolic taint propagation for StoreG statement not yet supported', 20: 'Symbolic taint propagation for LoadG statement not yet supported', 21: 'Symbolic taint propagation for CAS statement not yet supported', 22: 'Symbolic taint propagation for LLSC statement not yet supported', 23: 'Symbolic taint propagation for Dirty statement not yet supported', 24: 'Symbolic taint propagation for GetI expression not yet supported', 25: 'Canoo propagate symbolic taint for unsupported VEX statement type', 26: 'Cannot propagate symbolic taint for unsupported VEX expression', 27: 'Unicorn failed to determine size of memory write', 28: 'ARM syscalls are currently not supported by SimEngineUnicorn', 29: 'Block executes cpuid which should be handled in VEX engine'}

symbolic_stop_reasons = {12, 13, 14, 15, 16, 17, 28, 29}

unsupported_reasons = {11, 18, 19, 20, 21, 22, 23, 25, 26}

static name_stop(*num*)

static get_stop_msg(*stop_reason*)

class angr.state_plugins.unicorn_engine.StopDetails

Bases: Structure

struct stop_details_t

block_addr

Structure/Union member

block_size

Structure/Union member

stop_reason

Structure/Union member

class angr.state_plugins.unicorn_engine.SimOSEnum

Bases: object

enum simos_t

SIMOS_CGC = 0

SIMOS_LINUX = 1

SIMOS_OTHER = 2

exception angr.state_plugins.unicorn_engine.**MemoryMappingError**

Bases: Exception

exception angr.state_plugins.unicorn_engine.**AccessingZeroPageError**

Bases: *MemoryMappingError*

exception angr.state_plugins.unicorn_engine.**FetchingZeroPageError**

Bases: *MemoryMappingError*

exception angr.state_plugins.unicorn_engine.**SegfaultError**

Bases: *MemoryMappingError*

exception angr.state_plugins.unicorn_engine.**MixedPermissonsError**

Bases: *MemoryMappingError*

class angr.state_plugins.unicorn_engine.**AggressiveConcretizationAnnotation**

Bases: SimplificationAvoidanceAnnotation

**__init__**(*addr*)

class angr.state_plugins.unicorn_engine.**Uniwrapper**

Bases: object

Wrapper around unicorn.Uc that tracks mapped memory and hooks.

**__init__**(*arch*, *cache_key*, *thumb=False*)

**hook_add**(*htype*, *callback*, *user_data=None*, *begin=1*, *end=0*, *aux1=0*, *aux2=0*)

**hook_del**(*h*)

**mem_map**(*addr*, *size*, *perms=7*)

**mem_map_ptr**(*addr*, *size*, *perms*, *ptr*)

**mem_unmap**(*addr*, *size*)

**mem_reset**()

**hook_reset**()

**reset**()

class angr.state_plugins.unicorn_engine.**Unicorn**

Bases: *SimStatePlugin*

setup the unicorn engine for a state

**UC_CONFIG = {}**

**__init__**(*syscall_hooks=None*, *cache_key=None*, *unicount=None*, *symbolic_var_counts=None*, *symbolic_inst_counts=None*, *concretized_asts=None*, *always_concretize=None*, *never_concretize=None*, *concretize_at=None*, *concretization_threshold_memory=None*, *concretization_threshold_registers=None*, *concretization_threshold_instruction=None*, *cooldown_symbolic_stop=2*, *cooldown_unsupported_stop=2*, *cooldown_nonunicorn_blocks=100*, *cooldown_stop_point=1*, *max_steps=1000000*)

Initializes the Unicorn plugin for angr. This plugin handles communication with UnicornEngine.

**property uc**

**static delete_uc**()

**set_last_block_details**(*details*)

**set_stops**(*stop_points*)

**set_tracking**(*track_bbls*, *track_stack*)

**hook**()

**uncache_region**(*addr*, *length*)

**clear_page_cache**()

**setup**(*syscall_data=None*, *fd_bytes=None*)

**start**(*step=None*)

**get_recent_bbl_addrs**()

**get_stop_details**()

**finish**(*succ_state*)

**destroy**(*succ_state*)

**set_regs**()

setting unicorn registers

**setup_flags**()

**setup_gdt**(*fs*, *gs*)

**read_msr**(*msr=3221225728*)

**write_msr**(*val*, *msr=3221225728*)

**get_regs**(*succ_state*)

loading registers from unicorn. If succ_state is not None, update it instead of self.state. Needed when handling symbolic exits in native interface

### 10.43.30 angr.state_plugins.view

**class** angr.state_plugins.view.**SimRegNameView**

Bases: *SimStatePlugin*

**get**(*reg_name*)

**class** angr.state_plugins.view.**SimMemView**

Bases: *SimStatePlugin*

This is a convenient interface with which you can access a program's memory.

The interface works like this:

- You first use [array index notation] to specify the address you'd like to load from
- If at that address is a pointer, you may access the `deref` property to return a SimMemView at the address present in memory.

- You then specify a type for the data by simply accessing a property of that name. For a list of supported types, look at `state.mem.types`.
- You can then *refine* the type. Any type may support any refinement it likes. Right now the only refinements supported are that you may access any member of a struct by its member name, and you may index into a string or array to access that element.
- If the address you specified initially points to an array of that type, you can say *.array(n)* to view the data as an array of n elements.
- Finally, extract the structured data with `.resolved` or `.concrete`. `.resolved` will return bitvector values, while `.concrete` will return integer, string, array, etc values, whatever best represents the data.
- Alternately, you may store a value to memory, by assigning to the chain of properties that you've constructed. Note that because of the way python works, `x = s.mem[...].prop; x = val` will NOT work, you must say `s.mem[...].prop = val`.

For example:

```
>>> s.mem[0x601048].long
<long (64 bits) <BV64 0x4008d0> at 0x601048>
>>> s.mem[0x601048].long.resolved
<BV64 0x4008d0>
>>> s.mem[0x601048].deref
<<untyped> <unresolvable> at 0x4008d0>
>>> s.mem[0x601048].deref.string.concrete
'SOSNEAKY'
```

**__init__**(*ty=None*, *addr=None*, *state=None*)

types: ClassVar[dict] = {'CharT': char, 'DIR': struct DIR, 'FILE': struct FILE, 'FILE_t': struct FILE_t, '_Bool': bool, '_ENTRY': struct _ENTRY, '_IO_codecvt': struct _IO_codecvt, '_IO_iconv_t': struct _IO_iconv_t, '_IO_lock_t': struct pthread_mutex_t, '_IO_marker': struct _IO_marker, '_IO_wide_data': struct _IO_wide_data, '__action_fn_t': __action_fn_t, '__clock_t': uint32_t, '__dev_t': uint64_t, '__free_fn_t': __free_fn_t, '__ftw_func_t': __ftw_func_t, '__gid_t': unsigned int, '__ino64_t': unsigned long long, '__ino_t': unsigned long, '__int128': int128_t, '__int256': int256_t, '__int32': int, '__int64': long long, '__mbstate_t': struct __mbstate_t, '__mode_t': unsigned int, '__nlink_t': unsigned int, '__off64_t': long long, '__off_t': long, '__pid_t': int, '__suseconds_t': int64_t, '__time_t': long, '__uid_t': unsigned int, '_obstack_chunk': struct _obstack_chunk, 'aiocb': struct aiocb, 'aiocb64': struct aiocb64, 'aioinit': struct aioinit, 'argp': struct argp, 'argp_child': struct argp_child, 'argp_option': struct argp_option, 'argp_parser_t': (int, char *, struct argp_state*) -> int, 'argp_state': struct argp_state, 'basic_string': string_t, 'bool': bool, 'byte': uint8_t, 'cc_t': char, 'char': char, 'clock_t': uint32_t, 'comparison_fn_t': comparison_fn_t, 'crypt_data': struct crypt_data, 'dev_t': int, 'dirent': struct dirent, 'dirent64': struct dirent64, 'double': double, 'drand48_data': struct <anon>, 'dword': uint32_t, 'error_t': int, 'exit_status': struct exit_status, 'fd_set': struct fd_set, 'float': float, 'fpos64_t': struct fpos64_t, 'fpos_t': struct fpos_t, 'fstab': struct fstab, 'glob64_t': struct glob64_t, 'glob_t': struct glob_t, 'group': struct group, 'hostent': struct hostent, 'hsearch_data': struct hsearch_data, 'if_nameindex': struct if_nameindex, 'in_addr': struct in_addr, 'in_port_t': uint16_t, 'ino64_t': unsigned long long, 'ino_t': unsigned long, 'int': int, 'int16_t': int16_t, 'int32_t': int32_t, 'int64_t': int64_t, 'int8_t': int8_t, 'iovec': struct <anon>, 'itimerval': struct itimerval, 'lconv': struct lconv, 'long': long, 'long double': double, 'long int': long, 'long long': long long, 'long long int': long long, 'long signed': long, 'long unsigned int': unsigned long, 'mallinfo': struct mallinfo, 'mallinfo2': struct mallinfo2, 'mbstate_t': struct mbstate_t, 'mntent': struct mntent, 'mode_t': unsigned int, 'netent': struct netent, 'ntptimeval': struct ntptimeval, 'obstack': struct obstack, 'off64_t': long long, 'off_t': long, 'option': struct option, 'passwd': struct passwd, 'pid_t': int, 'printf_info': struct printf_info, 'protoent': struct protoent, 'ptrdiff_t': long, 'qword': uint64_t, 'random_data': struct <anon>, 'regex_t': struct regex_t, 'rlim64_t': uint64_t, 'rlim_t': unsigned long, 'rlimit': struct rlimit, 'rlimit64': struct rlimit64, 'rusage': struct rusage, 'sa_family_t': unsigned short, 'sched_param': struct sched_param, 'sem_t': int, 'sembuf': struct sembuf, 'servent': struct servent, 'sgttyb': struct sgttyb, 'short': short, 'short int': short, 'sigevent': struct sigevent, 'sighandler_t': sighandler_t, 'signed': int, 'signed char': char, 'signed int': int, 'signed long': long, 'signed long int': long, 'signed long long': long long, 'signed long long int': long long, 'signed short': short, 'signed short int': short, 'sigset_t': int, 'sigstack': struct sigstack, 'sigval': union sigval { sival_int int; sival_ptr void *; }, 'size_t': size_t, 'sockaddr': struct sockaddr, 'sockaddr_in': struct sockaddr_in, 'socklen_t': uint32_t, 'speed_t': long, 'ssize': size_t, 'ssize_t': size_t, 'std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>>': string_t, 'string': string_t, 'struct iovec': struct <anon>, 'struct stat': struct stat, 'struct stat64': struct stat64, 'struct timespec': struct timespec, 'struct timeval': struct timeval, 'tcflag_t': unsigned long, 'termios': struct termios, 'time_t': long, 'timespec': struct timeval, 'timeval': struct timeval, 'timex': struct timex, 'timezone': struct timezone, 'tm': struct tm, 'tms': struct tms, 'uint16_t': uint16_t, 'uint32_t': uint32_t, 'uint64_t': uint64_t, 'uint8_t': uint8_t, 'uintptr_t': unsigned long, 'unsigned': unsigned int, 'unsigned __int128': uint128_t, 'unsigned __int256': uint256_t, 'unsigned char': char, 'unsigned int': unsigned int, 'unsigned long': unsigned long, 'unsigned long int': unsigned long, 'unsigned long long': unsigned long long, 'unsigned long long int': unsigned long long, 'unsigned short': unsigned short, 'unsigned short int': unsigned short, 'utimbuf': struct utimbuf, 'utmp': struct utmp, 'utmpx': struct utmx, 'utsname': struct utsname, 'va_list': struct va_list[1], 'void': void,

**state:** *SimState*[Any, Any] = None

**struct:** *StructMode*

**with_type**(*sim_type*)

Returns a copy of the SimMemView with a type.

**Parameters**
**sim_type** (*SimType*) – The new type.

**Return type**
*SimMemView*

**Returns**
The typed SimMemView copy.

**property resolvable**

**property resolved**

**property concrete**

**property deref:** *SimMemView*

**array**(*n*)

**Return type**
*SimMemView*

**member**(*member_name*)

If self is a struct and member_name is a member of the struct, return that member element. Otherwise raise an exception.

**Return type**
*SimMemView*

**Parameters**
**member_name** (*str*)

**store**(*value*)

**class** angr.state_plugins.view.**StructMode**

Bases: object

**__init__**(*view*)

## 10.44 angr.storage

**class** angr.storage.**DefaultMemory**

Bases: *HexDumperMixin*, *SmartFindMixin*, *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *InspectMixin*, *ActionsMixinHigh*, *UnderconstrainedMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *AddressConcretizationMixin*, *ActionsMixinLow*, *ConditionalMixin*, *ConvenientMappingsMixin*, *DirtyAddrsMixin*, *StackAllocationMixin*, *ClemoryBackerMixin*, *DictBackerMixin*, *PrivilegedPagingMixin*, *UltraPagesMixin*, *DefaultFillerMixin*, *SymbolicMergerMixin*, *PagedMemoryMixin*

class angr.storage.SimFile

Bases: *SimFileBase*, *DefaultMemory*

The normal SimFile is meant to model files on disk. It subclasses SimSymbolicMemory so loads and stores to/from it are very simple.

**Parameters**

- **name** – The name of the file
- **content** – Optional initial content for the file as a string or bitvector
- **size** – Optional size of the file. If content is not specified, it defaults to zero
- **has_end** – Whether the size boundary is treated as the end of the file or a frontier at which new content will be generated. If unspecified, will pick its value based on options.FILES_HAVE_EOF. Another caveat is that if the size is also unspecified this value will default to False.
- **seekable** – Optional bool indicating whether seek operations on this file should succeed, default True.
- **writable** – Whether writing to this file is allowed
- **concrete** – Whether or not this file contains mostly concrete data. Will be used by some SimProcedures to choose how to handle variable-length operations like fgets.

**Variables**

**has_end** – Whether this file has an EOF

__init__(*name=None*, *content=None*, *size=None*, *has_end=None*, *seekable=True*, *writable=True*, *ident=None*, *concrete=None*, ***kwargs*)

property category

reg, mem, or file.

**Type**

Return the category of this SimMemory instance. It can be one of the three following categories

property size

The number of data bytes stored by the file at present. May be a symbolic value.

concretize(***kwargs*)

Return a concretization of the contents of the file, as a flat bytestring.

class angr.storage.SimMemoryObject

Bases: object

A SimMemoryObject is a reference to a byte or several bytes in a specific object in memory. It should be used only by the bottom layer of memory.

__init__(*obj*, *base*, *endness*, *length=None*, *byte_width=8*)

is_bytes

base

object: BV | FP

length

**endness**

**size**()

**property variables**

**property symbolic**

**property last_addr**

**concrete_bytes**(*offset*, *size*)

**Return type**
bytes | None

**Parameters**

- **offset** (*int*)
- **size** (*int*)

**includes**(*x*)

**bytes_at**(*addr*, *length*, *allow_concrete=False*, *endness='Iend_BE'*)

### Submodules


*file*
*memory_mixins*
*memory_object*


## 10.44.1 angr.storage.file

**class** angr.storage.file.**Flags**

Bases: object

**O_RDONLY = 0**

**O_WRONLY = 1**

**O_RDWR = 2**

**O_ACCMODE = 3**

**O_APPEND = 1024**

**O_ASYNC = 8192**

**O_CLOEXEC = 524288**

**O_CREAT = 64**

**O_DIRECT = 16384**

**O_DIRECTORY = 65536**

**O_DSYNC = 4096**

**O_EXCL = 128**

**O_LARGEFILE = 32768**

**O_NOATIME = 262144**

**O_NOCTTY = 256**

**O_NOFOLLOW = 131072**

**O_NONBLOCK = 2048**

**O_NDELAY = 2048**

**O_PATH = 2097152**

**O_SYNC = 1052672**

**O_TMPFILE = 4259840**

**O_TRUNC = 512**

**class** angr.storage.file.**SimFileBase**

Bases: *SimStatePlugin*

SimFiles are the storage mechanisms used by SimFileDescriptors.

Different types of SimFiles can have drastically different interfaces, and as a result there's not much that can be specified on this base class. All the read and write methods take a `pos` argument, which may have different semantics per-class. `0` will always be a valid position to use, though, and the next position you should use is part of the return tuple.

Some simfiles are "streams", meaning that the position that reads come from is determined not by the position you pass in (it will in fact be ignored), but by an internal variable. This is stored as `.pos` if you care to read it. Don't write to it. The same lack-of-semantics applies to this field as well.

**Variables**

- **name** – The name of the file. Purely for cosmetic purposes
- **ident** – The identifier of the file, typically autogenerated from the name and a nonce. Purely for cosmetic purposes, but does appear in symbolic values autogenerated in the file.
- **seekable** – Bool indicating whether seek operations on this file should succeed. If this is True, then `pos` must be a number of bytes from the start of the file.
- **writable** – Bool indicating whether writing to this file is allowed.
- **pos** – If the file is a stream, this will be the current position. Otherwise, None.
- **concrete** – Whether or not this file contains mostly concrete data. Will be used by some SimProcedures to choose how to handle variable-length operations like fgets.
- **file_exists** – Set to False, if file does not exists, set to a claripy Bool if unknown, default True.

**seekable = False**

**pos = None**

**__init__**(*name=None*, *writable=True*, *ident=None*, *concrete=False*, *file_exists=True*, ***kwargs*)

**static make_ident**(*name*)

**concretize**(***kwargs*)

Return a concretization of the contents of the file. The type of the return value of this method will vary depending on which kind of SimFile you're using.

**read**(*pos*, *size*, ***kwargs*)

Read some data from the file.

**Parameters**

- **pos** – The offset in the file to read from.
- **size** – The size to read. May be symbolic.

**Returns**

A tuple of the data read (a bitvector of the length that is the maximum length of the read), the actual size of the read, and the new file position pointer.

**write**(*pos*, *data*, *size=None*, ***kwargs*)

Write some data to the file.

**Parameters**

- **pos** – The offset in the file to write to. May be ignored if the file is a stream or device.
- **data** – The data to write as a bitvector
- **size** – The optional size of the data to write. If not provided will default to the length of the data. Must be constrained to less than or equal to the size of the data.

**Returns**

The new file position pointer.

**property size**

The number of data bytes stored by the file at present. May be a symbolic value.

**class** angr.storage.file.**SimFile**

Bases: *SimFileBase*, *DefaultMemory*

The normal SimFile is meant to model files on disk. It subclasses SimSymbolicMemory so loads and stores to/from it are very simple.

**Parameters**

- **name** – The name of the file
- **content** – Optional initial content for the file as a string or bitvector
- **size** – Optional size of the file. If content is not specified, it defaults to zero
- **has_end** – Whether the size boundary is treated as the end of the file or a frontier at which new content will be generated. If unspecified, will pick its value based on options.FILES_HAVE_EOF. Another caveat is that if the size is also unspecified this value will default to False.
- **seekable** – Optional bool indicating whether seek operations on this file should succeed, default True.
- **writable** – Whether writing to this file is allowed
- **concrete** – Whether or not this file contains mostly concrete data. Will be used by some SimProcedures to choose how to handle variable-length operations like fgets.

**Variables**

**has_end** – Whether this file has an EOF

__init__(*name=None*, *content=None*, *size=None*, *has_end=None*, *seekable=True*, *writable=True*, *ident=None*, *concrete=None*, ***kwargs*)

property category

reg, mem, or file.

**Type**
Return the category of this SimMemory instance. It can be one of the three following categories

property size

The number of data bytes stored by the file at present. May be a symbolic value.

concretize(***kwargs*)

Return a concretization of the contents of the file, as a flat bytestring.

class angr.storage.file.**SimFileStream**

Bases: *SimFile*

A specialized SimFile that uses a flat memory backing, but functions as a stream, tracking its position internally.

The pos argument to the read and write methods will be ignored, and will return None. Instead, there is an attribute pos on the file itself, which will give you what you want.

**Parameters**

- **name** – The name of the file, for cosmetic purposes
- **pos** – The initial position of the file, default zero
- **kwargs** – Any other keyword arguments will go on to the SimFile constructor.

**Variables**
**pos** – The current position in the file.

__init__(*name=None*, *content=None*, *pos=0*, ***kwargs*)

class angr.storage.file.**SimPackets**

Bases: *SimFileBase*

The SimPackets is meant to model inputs whose content is delivered a series of asynchronous chunks. The data is stored as a list of read or write results. For symbolic sizes, state.libc.max_packet_size will be respected. If the SHORT_READS option is enabled, reads will return a symbolic size constrained to be less than or equal to the requested size.

A SimPackets cannot be used for both reading and writing - for socket objects that can be both read and written to you should use a file descriptor to multiplex the read and write operations into two separate file storage mechanisms.

**Parameters**

- **name** – The name of the file, for cosmetic purposes
- **write_mode** – Whether this file is opened in read or write mode. If this is unspecified it will be autodetected.
- **content** – Some initial content to use for the file. Can be a list of bytestrings or a list of tuples of content ASTs and size ASTs.

**Variables**

- **write_mode** – See the eponymous parameter
- **content** – A list of packets, as tuples of content ASTs and size ASTs.

**__init__**(*name*, *write_mode=None*, *content=None*, *writable=True*, *ident=None*, ***kwargs*)

**property size**

The number of data bytes stored by the file at present. May be a symbolic value.

**concretize**(***kwargs*)

Returns a list of the packets read or written as bytestrings.

**read**(*pos*, *size*, ***kwargs*)

Read a packet from the stream.

**Parameters**

- **pos** (*int*) – The packet number to read from the sequence of the stream. May be None to append to the stream.
- **size** – The size to read. May be symbolic.
- **short_reads** – Whether to replace the size with a symbolic value constrained to less than or equal to the original size. If unspecified, will be chosen based on the state option.

**Returns**

A tuple of the data read (a bitvector of the length that is the maximum length of the read) and the actual size of the read.

**write**(*pos*, *data*, *size=None*, *events=True*, ***kwargs*)

Write a packet to the stream.

**Parameters**

- **pos** (*int*) – The packet number to write in the sequence of the stream. May be None to append to the stream.
- **data** – The data to write, as a string or bitvector.
- **size** – The optional size to write. May be symbolic; must be constrained to at most the size of data.

**Returns**

The next packet to use after this

**class** angr.storage.file.**SimPacketsStream**

Bases: *SimPackets*

A specialized SimPackets that tracks its position internally.

The pos argument to the read and write methods will be ignored, and will return None. Instead, there is an attribute `pos` on the file itself, which will give you what you want.

**Parameters**

- **name** – The name of the file, for cosmetic purposes
- **pos** – The initial position of the file, default zero
- **kwargs** – Any other keyword arguments will go on to the SimPackets constructor.

**Variables**

**pos** – The current position in the file.

**__init__**(*name*, *pos=0*, ***kwargs*)

class angr.storage.file.**SimFileDescriptorBase**

Bases: *SimStatePlugin*

The base class for implementations of POSIX file descriptors.

All file descriptors should respect the CONCRETIZE_SYMBOLIC_{READ,WRITE}_SIZES state options.

**read**(*pos*, *size*, ***kwargs*)

Reads some data from the file, storing it into memory.

**Parameters**

- **pos** – The address to read data from file
- **size** – The requested length of the read

**Returns**
The real length of the read

**write**(*pos*, *size*, ***kwargs*)

Writes some data, loaded from the state, into the file.

**Parameters**

- **pos** – The address to read the data to write from in memory
- **size** – The requested size of the write

**Returns**
The real length of the write

**read_data**(*size*, ***kwargs*)

Reads some data from the file, returning the data.

**Parameters**
**size** – The requested length of the read

**Returns**
A tuple of the data read and the real length of the read

**write_data**(*data*, *size=None*, ***kwargs*)

Write some data, provided as an argument into the file.

**Parameters**

- **data** – A bitvector to write into the file
- **size** – The requested size of the write (may be symbolic)

**Returns**
The real length of the write

**seek**(*offset*, *whence='start'*)

Seek the file descriptor to a different position in the file.

**Parameters**

- **offset** – The offset to seek to, interpreted according to whence
- **whence** – What the offset is relative to; one of the strings “start”, “current”, or “end”

**Returns**
A symbolic boolean describing whether the seek succeeded or not

**tell**()

Return the current position, or None if the concept doesn't make sense for the given file.

**eof**()

Return the EOF status. May be a symbolic boolean.

**size**()

Return the size of the data stored in the file in bytes, or None if the concept doesn't make sense for the given file.

**property read_storage**

Return the SimFile backing reads from this fd

**property write_storage**

Return the SimFile backing writes to this fd

**property read_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**property write_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**concretize**(***kwargs*)

Return a concretizeation of the data in the underlying file. Has different return types to represent different data structures on a per-class basis.

Any arguments passed to this will be passed onto state.solver.eval.

**property file_exists**

This should be True in most cases. Only if we opened an fd of unknown existence, ALL_FILES_EXIST is False and ANY_FILE_MIGHT_EXIST is True, this is a symbolic boolean.

**class** angr.storage.file.**SimFileDescriptor**

Bases: *SimFileDescriptorBase*

A simple file descriptor forwarding reads and writes to a SimFile. Contains information about the current opened state of the file, such as the flags or (if relevant) the current position.

**Variables**

- **file** – The SimFile described to by this descriptor
- **flags** – The mode that the file descriptor was opened with, a bitfield of flags

**__init__**(*simfile*, *flags=0*)

**concretize**(***kwargs*)

Return a concretization of the underlying file. Returns whatever format is preferred by the file.

**property file_exists**

This should be True in most cases. Only if we opened an fd of unknown existence, ALL_FILES_EXIST is False and ANY_FILE_MIGHT_EXIST is True, this is a symbolic boolean.

**property read_storage**

Return the SimFile backing reads from this fd

**property write_storage**

Return the SimFile backing writes to this fd

**property read_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**property write_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**class** angr.storage.file.**SimFileDescriptorDuplex**

Bases: *SimFileDescriptorBase*

A file descriptor that refers to two file storage mechanisms, one to read from and one to write to. As a result, operations like seek, eof, etc no longer make sense.

**Parameters**

- **read_file** – The SimFile to read from
- **write_file** – The SimFile to write to

**__init__**(*read_file*, *write_file*)

**concretize**(***kwargs*)

Return a concretization of the underlying files, as a tuple of (read file, write file).

**property read_storage**

Return the SimFile backing reads from this fd

**property write_storage**

Return the SimFile backing writes to this fd

**property read_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**property write_pos**

Return the current position of the read file pointer.

If the underlying read file is a stream, this will return the position of the stream. Otherwise, will return the position of the file descriptor in the file.

**class** angr.storage.file.**SimPacketsSlots**

Bases: *SimFileBase*

SimPacketsSlots is the new SimDialogue, if you've ever seen that before.

The idea is that in some cases, the only thing you really care about is getting the lengths of reads right, and some of them should be short reads, and some of them should be truncated. You provide to this class a list of read lengths, and it figures out the length of each read, and delivers some content.

This class will NOT respect the position argument you pass it - this storage is not stateless.

**seekable = False**

**__init__**(*name*, *read_sizes*, *ident=None*, ***kwargs*)

**property size**

The number of data bytes stored by the file at present. May be a symbolic value.

### 10.44.2 angr.storage.memory_mixins

**class** angr.storage.memory_mixins.**AbstractMemory**

Bases: *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *InspectMixin*, *ActionsMixinHigh*, *UnderconstrainedMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *ActionsMixinLow*, *ConditionalMixin*, *RegionedAddressConcretizationMixin*, *RegionedMemoryMixin*

**class** angr.storage.memory_mixins.**AbstractMergerMixin**

Bases: *MemoryMixin*

AbstractMergerMixin handles merging initialized values.

**class** angr.storage.memory_mixins.**ActionsMixinHigh**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, ***, *condition=None*, *fallback=None*, *disable_actions=False*, *action=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *disable_actions=False*, *action=None*, *condition=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**ActionsMixinLow**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, ***, *action=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *action=None*, ***kwargs*)

**Parameters**
**action** (SimActionData | *None*)

**class** angr.storage.memory_mixins.**AddressConcretizationMixin**

Bases: *MemoryMixin*

The address concretization mixin allows symbolic reads and writes to be handled sanely by dispatching them as a number of conditional concrete reads/writes. It provides a "concretization strategies" interface allowing the process of serializing symbolic addresses into concrete ones to be specified.

**__init__**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

**concretize_write_addr**(*addr*, *strategies=None*, *condition=None*)

Concretizes an address meant for writing.

**Parameters**

- **addr** – An expression for the address.
- **strategies** – A list of concretization strategies (to override the default).
- **condition** – Any extra constraints that should be observed when determining address satisfiability

**Returns**
A list of concrete addresses.

**concretize_read_addr**(*addr*, *strategies=None*, *condition=None*)

Concretizes an address meant for reading.

**Parameters**

- **addr** – An expression for the address.
- **strategies** – A list of concretization strategies (to override the default).

**Returns**

A list of concrete addresses.

**load**(*addr*, *size=None*, *, *condition=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, *, *condition=None*, ***kwargs*)

**permissions**(*addr*, *permissions=None*, ***kwargs*)

**map_region**(*addr*, *length*, *permissions*, ***kwargs*)

**unmap_region**(*addr*, *length*, ***kwargs*)

**class** angr.storage.memory_mixins.**ClemoryBackerMixin**

Bases: *PagedMemoryMixin*

**__init__**(*cle_memory_backer=None*, ***kwargs*)

**Parameters**

**cle_memory_backer** (*Loader | Clemory | None*)

**class** angr.storage.memory_mixins.**ConditionalMixin**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, *, *condition=None*, *fallback=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, *, *condition=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**ConvenientMappingsMixin**

Bases: *MemoryMixin*

Implements mappings between names and hashes of symbolic variables and these variables themselves.

**__init__**(***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**get_symbolic_addrs**()

**addrs_for_name**(*n*)

Returns addresses that contain expressions that contain a variable named *n*.

**addrs_for_hash**(*h*)

Returns addresses that contain expressions that contain a variable with the hash of *h*.

**replace_all**(*old*, *new*)

Replaces all instances of expression *old* with expression *new*.

**Parameters**

- **old** (BV) – A claripy expression. Must contain at least one named variable (to make it possible to use the name index for speedup).
- **new** (BV) – The new variable to replace it with.

**class** angr.storage.memory_mixins.**CooperationBase**

Bases: Generic

Any given subclass of this class which is not a subclass of MemoryMixin should have the property that any subclass it which *is* a subclass of MemoryMixin should all work with the same datatypes

**class** angr.storage.memory_mixins.**DataNormalizationMixin**

Bases: *MemoryMixin*

Normalizes the data field for a store and the fallback field for a load to be BVs.

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *fallback=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**DefaultFillerMixin**

Bases: *MemoryMixin*

**class** angr.storage.memory_mixins.**DefaultListPagesMemory**

Bases: *HexDumperMixin*, *SmartFindMixin*, *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *ActionsMixinHigh*, *UnderconstrainedMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *InspectMixin*, *AddressConcretizationMixin*, *ActionsMixinLow*, *ConditionalMixin*, *ConvenientMappingsMixin*, *DirtyAddrsMixin*, *StackAllocationMixin*, *ClemoryBackerMixin*, *DictBackerMixin*, *PrivilegedPagingMixin*, *ListPagesMixin*, *DefaultFillerMixin*, *SymbolicMergerMixin*, *PagedMemoryMixin*

**class** angr.storage.memory_mixins.**DefaultMemory**

Bases: *HexDumperMixin*, *SmartFindMixin*, *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *InspectMixin*, *ActionsMixinHigh*, *UnderconstrainedMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *AddressConcretizationMixin*, *ActionsMixinLow*, *ConditionalMixin*, *ConvenientMappingsMixin*, *DirtyAddrsMixin*, *StackAllocationMixin*, *ClemoryBackerMixin*, *DictBackerMixin*, *PrivilegedPagingMixin*, *UltraPagesMixin*, *DefaultFillerMixin*, *SymbolicMergerMixin*, *PagedMemoryMixin*

**class** angr.storage.memory_mixins.**DictBackerMixin**

Bases: *PagedMemoryMixin*

**__init__**(*dict_memory_backer=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**DirtyAddrsMixin**

Bases: *MemoryMixin*

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**ExplicitFillerMixin**

Bases: *MemoryMixin*

**__init__**(*uninitialized_read_handler=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**FastMemory**

Bases: *NameResolutionMixin*, *SimpleInterfaceMixin*, *SimplificationMixin*, *InspectMixin*, *ConditionalMixin*, *ExplicitFillerMixin*, *DefaultFillerMixin*, *SlottedMemoryMixin*

**class** angr.storage.memory_mixins.**HexDumperMixin**

Bases: *MemoryMixin*

**hex_dump**(*start*, *size*, *word_size=4*, *words_per_row=4*, *endianness='Iend_BE'*, *symbolic_char='?'*, *unprintable_char='.'*, *solve=False*, *extra_constraints=None*, *inspect=False*, *disable_actions=True*)

Returns a hex dump as a string. The solver, if enabled, is called once for every byte potentially making this function very slow. It is meant to be used mainly as a "visualization" for debugging.

Warning: May read and display more bytes than *size* due to rounding. Particularly, if size is less than, or not a multiple of word_size*words_per_line.

**Parameters**

- **start** – starting address from which to print
- **size** – number of bytes to display
- **word_size** – number of bytes to group together as one space-delimited unit
- **words_per_row** – number of words to display per row of output
- **endianness** – endianness to use when displaying each word (ASCII representation is unchanged)
- **symbolic_char** – the character to display when a byte is symbolic and has multiple solutions
- **unprintable_char** – the character to display when a byte is not printable
- **solve** – whether or not to attempt to solve (warning: can be very slow)
- **extra_constraints** – extra constraints to pass to the solver is solve is True
- **inspect** – whether or not to trigger SimInspect breakpoints for the memory load
- **disable_actions** – whether or not to disable SimActions for the memory load

**Returns**
hex dump as a string

**class** angr.storage.memory_mixins.**HistoryTrackingMixin**

Bases: *RefcountMixin*, *MemoryMixin*

Tracks the history of memory writes.

**__init__**(**args*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**parents**()

**changed_bytes**(*other*, ***kwargs*)

**Return type**
set[int] | None

**all_bytes_changed_in_history**()

**Return type**
SegmentList

**class** angr.storage.memory_mixins.**ISPOMixin**

Bases: *MemoryMixin*

An implementation of the International Stateless Persons Organisation, a mixin which should be applied as a bottom layer for memories which have no state and must redirect certain operations to a parent memory. Main usecase is for memory region classes which are stored within other memories, such as pages.

class angr.storage.memory_mixins.**InspectMixin**

Bases: *MemoryMixin*

This mixin adds support for the inspect plugin to the memory model.

Inspect breakpoints are called for the following events:

- mem_read (before and after)
- mem_write (before and after)
- reg_read (before and after)
- reg_write (before and after)
- address_concretization_add_constraints (before)

**store**(*addr*, *data*, *size=None*, ***, *condition=None*, *endness=None*, *inspect=True*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *condition=None*, *endness=None*, *inspect=True*, ***kwargs*)

class angr.storage.memory_mixins.**JavaVmMemory**

Bases: *JavaVmMemoryMixin*

class angr.storage.memory_mixins.**JavaVmMemoryMixin**

Bases: *MemoryMixin*

A memory mixin for JavaVM memory.

**__init__**(*memory_id='mem'*, *stack=None*, *heap=None*, *vm_static_table=None*, *load_strategies=None*, *store_strategies=None*, *max_array_size=1000*, ***kwargs*)

**static get_new_uuid**()

Generate a unique id within the scope of the JavaVM memory. This, for example, is used for distinguishing memory objects of the same type (e.g. multiple instances of the same class).

**store**(*addr*, *data*, *frame=0*)

**load**(*addr*, *frame=0*, *none_if_missing=False*)

**push_stack_frame**()

**pop_stack_frame**()

**property stack**

**store_array_element**(*array*, *idx*, *value*)

**store_array_elements**(*array*, *start_idx*, *data*)

Stores either a single element or a range of elements in the array.

**Parameters**

- **array** – Reference to the array.
- **start_idx** – Starting index for the store.
- **data** – Either a single value or a list of values.

**load_array_element**(*array*, *idx*)

**load_array_elements**(*array*, *start_idx*, *no_of_elements*)

Loads either a single element or a range of elements from the array.

**Parameters**

- **array** – Reference to the array.
- **start_idx** – Starting index for the load.
- **no_of_elements** – Number of elements to load.

**concretize_store_idx**(*idx*, *strategies=None*)

Concretizes a store index.

**Parameters**

- **idx** – An expression for the index.
- **strategies** – A list of concretization strategies (to override the default).
- **min_idx** – Minimum value for a concretized index (inclusive).
- **max_idx** – Maximum value for a concretized index (exclusive).

**Returns**

A list of concrete indexes.

**concretize_load_idx**(*idx*, *strategies=None*)

Concretizes a load index.

**Parameters**

- **idx** – An expression for the index.
- **strategies** – A list of concretization strategies (to override the default).
- **min_idx** – Minimum value for a concretized index (inclusive).
- **max_idx** – Maximum value for a concretized index (exclusive).

**Returns**

A list of concrete indexes.

**class** angr.storage.memory_mixins.**KeyValueMemory**

Bases: *KeyValueMemoryMixin*

**class** angr.storage.memory_mixins.**KeyValueMemoryMixin**

Bases: *MemoryMixin*

KeyValueMemoryMixin is a mixin that provides a simple key-value store for memory.

**__init__**(**args*, ***kwargs*)

**load**(*addr*, *size=None*, *none_if_missing=False*, ***kwargs*)

**store**(*addr*, *data*, *type_=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**LabelMergerMixin**

Bases: *MemoryMixin*

A memory mixin for merging labels. Labels come from SimLabeledMemoryObject.

**__init__**(**args*, ***kwargs*)

class angr.storage.memory_mixins.**LabeledMemory**

Bases: *SizeNormalizationMixin*, *ListPagesWithLabelsMixin*, *DefaultFillerMixin*, *TopMergerMixin*, *LabelMergerMixin*, *PagedMemoryMixin*

LabeledMemory is used in static analysis. It allows storing values with labels, such as *Definition*.

class angr.storage.memory_mixins.**ListPage**

Bases: *MemoryObjectMixin*, *PageBase*

This class implements a page memory mixin with lists as the main content store.

**__init__**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

**load**(*addr*, *size=None*, *endness=None*, *page_addr=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**changed_bytes**(*other*, *page_addr=None*)

**Parameters**

- **other** (ListPage)
- **page_addr** (*int | None*)

class angr.storage.memory_mixins.**ListPagesMixin**

Bases: *PagedMemoryMixin*

**PAGE_TYPE**

alias of *ListPage*

class angr.storage.memory_mixins.**ListPagesWithLabelsMixin**

Bases: *LabeledPagesMixin*, *ListPagesMixin*

class angr.storage.memory_mixins.**MVListPage**

Bases: *MemoryObjectSetMixin*, *PageBase*

MVListPage allows storing multiple values at the same location.

Each store() may take a value or multiple values. Each load() returns an iterator of all values stored at that location.

**__init__**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

**load**(*addr*, *size=None*, *endness=None*, *page_addr=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**Return type**
list[tuple[int, *SimMemoryObject* | *SimLabeledMemoryObject*]]

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**compare**(*other*, *page_addr=None*, *memory=None*, *changed_offsets=None*)

**Return type**
bool

**Parameters**

- **other** (MVListPage)
- **page_addr** (*int | None*)

**changed_bytes**(*other*, *page_addr=None*)

**Parameters**

- **other** (MVListPage)
- **page_addr** (*int | None*)

**content_gen**(*index*)

class angr.storage.memory_mixins.**MVListPagesMixin**

Bases: *PagedMemoryMixin*

**PAGE_TYPE**

alias of *MVListPage*

**__init__**(**args*, *skip_missing_values_during_merging=False*, ***kwargs*)

class angr.storage.memory_mixins.**MVListPagesWithLabelsMixin**

Bases: *LabeledPagesMixin*, *MVListPagesMixin*

class angr.storage.memory_mixins.**MemoryObjectMixin**

Bases: *CooperationBase*[*SimMemoryObject*]

Uses SimMemoryObjects in region storage. With this, load will return a list of tuple (address, MO) and store will take a MO.

class angr.storage.memory_mixins.**MemoryRegionMetaMixin**

Bases: *MemoryMixin*

**__init__**(*related_function_addr=None*, ***kwargs*)

**property is_stack**

**property related_function_addr**

**get_abstract_locations**(*addr*, *size*)

Get a list of abstract locations that is within the range of [addr, addr + size]

This implementation is pretty slow. But since this method won't be called frequently, we can live with the bad implementation for now.

**Parameters**

- **addr** – Starting address of the memory region.
- **size** – Size of the memory region, in bytes.

**Returns**

A list of covered AbstractLocation objects, or an empty list if there is none.

**store**(*addr*, *data*, *size=None*, ***, *bbl_addr=None*, *stmt_id=None*, *ins_addr=None*, *endness=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *bbl_addr=None*, *stmt_idx=None*, *ins_addr=None*, ***kwargs*)

**dbg_print**(*indent=0*)

Print out debugging information

class angr.storage.memory_mixins.**MultiValueMergerMixin**

Bases: *MemoryMixin*

__init__(**args*, *element_limit=5*, *annotation_limit=256*, *top_func=None*, *is_top_func=None*, *phi_maker=None*, *merge_into_top=True*, ***kwargs*)

class angr.storage.memory_mixins.**MultiValuedMemory**

Bases: *SizeNormalizationMixin*, *MVListPagesMixin*, *DefaultFillerMixin*, *MultiValueMergerMixin*, *PagedMemoryMixin*, *PagedMemoryMultiValueMixin*

class angr.storage.memory_mixins.**NameResolutionMixin**

Bases: *MemoryMixin*

This mixin allows you to provide register names as load addresses, and will automatically translate this to an offset and size.

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***kwargs*)

class angr.storage.memory_mixins.**PageBase**

Bases: *HistoryTrackingMixin*, *RefcountMixin*, *CooperationBase*, *ISPOMixin*, *PermissionsMixin*, *MemoryMixin*

This is a fairly succinct definition of the contract between PagedMemoryMixin and its constituent pages:

- Pages must implement the MemoryMixin model for loads, stores, copying, merging, etc
- However, loading/storing may not necessarily use the same data domain as PagedMemoryMixin. In order to do more efficient loads/stores across pages, we use the CooperationBase interface which allows the page class to determine how to generate and unwrap the objects which are actually stored.
- To support COW, we use the RefcountMixin and the ISPOMixin (which adds the contract element that `memory=self` be passed to every method call)
- Pages have permissions associated with them, stored in the PermissionsMixin.

Read the docstrings for each of the constituent classes to understand the nuances of their functionalities

class angr.storage.memory_mixins.**PagedMemoryMixin**

Bases: *MemoryMixin*[int | BV | *SimActionObject*, BV, int | BV | *SimActionObject*], Generic

A bottom-level storage mechanism. Dispatches reads to individual pages, the type of which is the PAGE_TYPE class variable.

**SUPPORTS_CONCRETE_LOAD: bool = True**

**PAGE_TYPE: *type*[PageType]**

__init__(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *endness=None*, ***kwargs*)

**Parameters**

- **addr** (*int*)
- **size** (*int | None*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

**Parameters**

- **addr** (*int*)

- **size** (*int | None*)

**compare**(*other*)

**Return type**
bool

**Parameters**
**other** (PagedMemoryMixin)

**permissions**(*addr*, *permissions=None*, ***kwargs*)

**map_region**(*addr*, *length*, *permissions*, ***, *init_zero=False*, ***kwargs*)

**unmap_region**(*addr*, *length*, ***kwargs*)

**changed_bytes**(*other*)

**Return type**
set[int]

**changed_pages**(*other*)

**Return type**
dict[int, set[int] | None]

**flush_pages**(*white_list*)

Flush all pages not included in the *white_list* by removing their pages. Note, this will not wipe them from memory if they were backed by a memory_backer, it will simply reset them to their initial state. Returns the list of pages that were cleared consisting of *(addr, length)* tuples. :type white_list: :param white_list: white list of regions in the form of (start, end) to exclude from the flush :return: a list of memory page ranges that were flushed :rtype: list

**class** angr.storage.memory_mixins.**PagedMemoryMultiValueMixin**

Bases: *MemoryMixin*

Implement optimizations and fast accessors for the MultiValues-variant of Paged Memory.

**load_annotations**(*addr*, *size*, ***kwargs*)

**Parameters**

- **addr** (*int*)
- **size** (*int*)

**class** angr.storage.memory_mixins.**PermissionsMixin**

Bases: *MemoryMixin*

This mixin adds a permissions_bits field and properties for extracting the read/write/exec permissions. It does NOT add permissions checking.

**__init__**(*permissions=None*, ***kwargs*)

**Parameters**
**permissions** (*int | BV | None*)

**property perm_read**

**property perm_write**

**property perm_exec**

class angr.storage.memory_mixins.**PrivilegedPagingMixin**

Bases: *PagedMemoryMixin*

A mixin for paged memory models which will raise SimSegfaultExceptions if STRICT_PAGE_ACCESS is enabled and a segfault condition is detected.

Segfault conditions include: - getting a page for reading which is non-readable - getting a page for writing which is non-writable - creating a page

The latter condition means that this should be inserted under any mixins which provide other implementations of `_initialize_page`.

class angr.storage.memory_mixins.**RefcountMixin**

Bases: *MemoryMixin*

This mixin adds a locked reference counter and methods to manipulate it, to facilitate copy-on-write optimizations.

**__init__**(***kwargs*)

**acquire_unique**()

Call this function to return a version of this page which can be used for writing, which may or may not be the same object as before. If you use this you must immediately replace the shared reference you previously had with the new unique copy.

**acquire_shared**()

Call this function to indicate that this page has had a reference added to it and must be copied before it can be acquired uniquely again. Creating the object implicitly starts it with one shared reference.

**Return type**
None

**release_shared**()

Call this function to indicate that this page has had a shared reference to it released

**Return type**
None

class angr.storage.memory_mixins.**RegionCategoryMixin**

Bases: *MemoryMixin*

**property category**

reg, mem, or file.

**Type**
Return the category of this SimMemory instance. It can be one of the three following categories

class angr.storage.memory_mixins.**RegionedAddressConcretizationMixin**

Bases: *MemoryMixin*

**__init__**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

class angr.storage.memory_mixins.**RegionedMemory**

Bases: *RegionCategoryMixin*, *MemoryRegionMetaMixin*, *StaticFindMixin*, *UnwrapperMixin*, *NameResolutionMixin*, *DataNormalizationMixin*, *SimplificationMixin*, *SizeConcretizationMixin*, *SizeNormalizationMixin*, *AddressConcretizationMixin*, *ConvenientMappingsMixin*, *DirtyAddrsMixin*, *ClemoryBackerMixin*, *DictBackerMixin*, *UltraPagesMixin*, *DefaultFillerMixin*, *AbstractMergerMixin*, *PagedMemoryMixin*

class angr.storage.memory_mixins.**RegionedMemoryMixin**

Bases: *MemoryMixin*

Regioned memory. This mixin manages multiple memory regions. Each address is represented as a tuple of (region ID, offset into the region), which is called a regioned address.

Converting absolute addresses into regioned addresses: We map an absolute address to a region by looking up which region this address belongs to in the region map. Currently this is only enabled for stack. Heap support has not landed yet.

When start analyzing a function, the user should call set_stack_address_mapping() to create a new region mapping. Likewise, when exiting from a function, the user should cancel the previous mapping by calling unset_stack_address_mapping().

**__init__**(*write_targets_limit=2048*, *read_targets_limit=4096*, *stack_region_map=None*, *generic_region_map=None*, *stack_size=65536*, *cle_memory_backer=None*, *dict_memory_backer=None*, *regioned_memory_cls=None*, ***kwargs*)

**Parameters**

- **write_targets_limit** (*int*)
- **read_targets_limit** (*int*)
- **stack_region_map** (RegionMap | *None*)
- **generic_region_map** (RegionMap | *None*)
- **stack_size** (*int*)
- **cle_memory_backer** (*None*)
- **dict_memory_backer** (*dict* | *None*)
- **regioned_memory_cls** (type | *None*)

**load**(*addr*, *size=None*, ***, *endness=None*, *condition=None*, ***kwargs*)

**Parameters**

- **size** (*BV* | *int* | *None*)
- **condition** (*Bool* | *None*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

**Parameters**
**size** (*int* | *None*)

**find**(*addr*, *data*, *max_search*, ***kwargs*)

**Parameters**
**addr** (*int* | *Bits*)

**replace_all**(*old*, *new*)

**Parameters**

- **old** (*BV*)
- **new** (*BV*)

**set_stack_address_mapping**(*absolute_address*, *region_id*, *related_function_address=None*)

Create a new mapping between an absolute address (which is the base address of a specific stack frame) and a region ID.

**Parameters**

- **absolute_address** (int) – The absolute memory address.
- **region_id** (str) – The region ID.
- **related_function_address** (int | None) – Related function address.

**unset_stack_address_mapping**(*absolute_address*)

Remove a stack mapping.

**Parameters**
**absolute_address** (int) – An absolute memory address that is the base address of the stack frame to destroy.

**stack_id**(*function_address*)

Return a memory region ID for a function. If the default region ID exists in the region mapping, an integer will appended to the region name. In this way we can handle recursive function calls, or a function that appears more than once in the call frame.

This also means that *stack_id()* should only be called when creating a new stack frame for a function. You are not supposed to call this function every time you want to map a function address to a stack ID.

**Parameters**
**function_address** (int) – Address of the function.

**Return type**
str

**Returns**
ID of the new memory region.

**set_stack_size**(*size*)

**Parameters**
**size** (*int*)

**class** angr.storage.memory_mixins.**SimpleInterfaceMixin**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, ***, *endness=None*, *condition=None*, *fallback=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, *condition=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**SimplificationMixin**

Bases: *MemoryMixin*

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**SizeConcretizationMixin**

Bases: *MemoryMixin*

This mixin allows memory to process symbolic sizes. It will not touch any sizes which are not ASTs with non-BVV ops. Assumes that the data is a BV.

- symbolic load sizes will be concretized as their maximum and a warning will be logged
- symbolic store sizes will be dispatched as several conditional stores with concrete sizes

**__init__**(*concretize_symbolic_write_size=False*, *max_concretize_count=256*, *max_symbolic_size=4194304*, *raise_memory_limit_error=False*, *size_limit=257*, ***kwargs*)

**Parameters**

- **concretize_symbolic_write_size** (*bool*)
- **max_concretize_count** (*int | None*)
- **max_symbolic_size** (*int*)
- **raise_memory_limit_error** (*bool*)
- **size_limit** (*int*)

**load**(*addr*, *size=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**SizeNormalizationMixin**

Bases: *MemoryMixin*

Provides basic services related to normalizing sizes. After this mixin, sizes will always be a plain int. Assumes that the data is a BV.

- load will throw a TypeError if no size is provided
- store will default to len(data)//byte_width if no size is provided

**load**(*addr*, *size=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**SlottedMemoryMixin**

Bases: *MemoryMixin*

**__init__**(*width=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *endness=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

**changed_bytes**(*other*)

**class** angr.storage.memory_mixins.**SmartFindMixin**

Bases: *MemoryMixin*

Memory mixin providing basic searching over concrete and symbolic data.

**find**(*addr*, *data*, *max_search*, ***, *default=None*, *endness=None*, *chunk_size=None*, *max_symbolic_bytes=None*, *condition=None*, *char_size=1*, ***kwargs*)

**class** angr.storage.memory_mixins.**SpecialFillerMixin**

Bases: *MemoryMixin*

**__init__**(*special_memory_filler=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**StackAllocationMixin**

Bases: *PagedMemoryMixin*

This mixin adds automatic allocation for a stack region based on the stack_end and stack_size parameters.

**__init__**(*stack_end=None*, *stack_size=None*, *stack_perms=None*, ***kwargs*)

**allocate_stack_pages**(*addr*, *size*, ***kwargs*)

Pre-allocates pages for the stack without triggering any logic related to reading from them.

**Parameters**

- **addr** (int) – The highest address that should be mapped
- **size** (int) – The number of bytes to be allocated. byte 1 is the one at addr, byte 2 is the one before that, and so on.

**Returns**
A list of the new page objects

**class** angr.storage.memory_mixins.**StaticFindMixin**

Bases: *SmartFindMixin*

Implements data finding for abstract memory.

**find**(*addr*, *data*, *max_search*, ***, *default=None*, *endness=None*, *chunk_size=None*, *max_symbolic_bytes=None*, *condition=None*, *char_size=1*, ***kwargs*)

**class** angr.storage.memory_mixins.**SymbolicMergerMixin**

Bases: *MemoryMixin*

**class** angr.storage.memory_mixins.**TopMergerMixin**

Bases: *MemoryMixin*

A memory mixin for merging values in memory to TOP.

**__init__**(**args*, *top_func=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**UltraPage**

Bases: *MemoryObjectMixin*, *PageBase*

Default page implementation

**SUPPORTS_CONCRETE_LOAD: bool = True**

**__init__**(*memory=None*, *init_zero=False*, ***kwargs*)

**classmethod new_from_shared**(*data*, *memory=None*, ***kwargs*)

**load**(*addr*, *size=None*, *page_addr=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *page_addr=None*, *cooperate=False*, ***kwargs*)

**Parameters**

- **data** (*int | SimMemoryObject*)
- **size** (*int | None*)

**concrete_run_length**(*addr*, *size*, ***kwargs*)

Return the number of concrete bytes at addr, capped at size.

**Return type**
int

**changed_bytes**(*other*, *page_addr=None*)

**Return type**
set[int]

**replace_all_with_offsets**(*offsets*, *old*, *new*, *memory=None*)

**Parameters**

- **offsets** (*Iterable[int]*)
- **old** (*BV*)
- **new** (*BV*)

**class** angr.storage.memory_mixins.**UltraPagesMixin**

Bases: *PagedMemoryMixin*

**PAGE_TYPE**

alias of *UltraPage*

**class** angr.storage.memory_mixins.**UnderconstrainedMixin**

Bases: *MemoryMixin*

**__init__**(**args*, ***kwargs*)

**load**(*addr*, *size=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**class** angr.storage.memory_mixins.**UnwrapperMixin**

Bases: *MemoryMixin*

This mixin processes SimActionObjects by passing on their .ast field.

**store**(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *condition=None*, *fallback=None*, ***kwargs*)

**find**(*addr*, *data*, *max_search*, ***, *default=None*, ***kwargs*)

## Submodules

| |
|---|
| *actions_mixin* |
| *address_concretization_mixin* |
| *bvv_conversion_mixin* |
| *conditional_store_mixin* |
| *convenient_mappings_mixin* |
| *default_filler_mixin* |
| *dirty_addrs_mixin* |
| *hex_dumper_mixin* |
| *inspect_mixin* |
| *javavm_memory_mixin* |
| *keyvalue_memory_mixin* |
| *label_merger_mixin* |
| *memory_mixin* |
| *multi_value_merger_mixin* |
| *name_resolution_mixin* |
| *paged_memory* |
| *regioned_memory* |
| *simple_interface_mixin* |
| *simplification_mixin* |

continues on next page

Table 111 – continued from previous page

| |
|---|
| *size_resolution_mixin* |
| *slotted_memory* |
| *smart_find_mixin* |
| *symbolic_merger_mixin* |
| *top_merger_mixin* |
| *underconstrained_mixin* |
| *unwrapper_mixin* |

### angr.storage.memory_mixins.actions_mixin

**class** angr.storage.memory_mixins.actions_mixin.**ActionsMixinHigh**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, ***, *condition=None*, *fallback=None*, *disable_actions=False*, *action=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *disable_actions=False*, *action=None*, *condition=None*, ***kwargs*)

**class** angr.storage.memory_mixins.actions_mixin.**ActionsMixinLow**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, ***, *action=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *action=None*, ***kwargs*)

**Parameters**

**action** (SimActionData | *None*)

### angr.storage.memory_mixins.address_concretization_mixin

**class** angr.storage.memory_mixins.address_concretization_mixin.**MultiwriteAnnotation**

Bases: Annotation

**eliminatable = False**

**relocatable = True**

**class** angr.storage.memory_mixins.address_concretization_mixin.**AddressConcretizationMixin**

Bases: *MemoryMixin*

The address concretization mixin allows symbolic reads and writes to be handled sanely by dispatching them as a number of conditional concrete reads/writes. It provides a "concretization strategies" interface allowing the process of serializing symbolic addresses into concrete ones to be specified.

**__init__**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

**concretize_write_addr**(*addr*, *strategies=None*, *condition=None*)

Concretizes an address meant for writing.

**Parameters**

- **addr** – An expression for the address.
- **strategies** – A list of concretization strategies (to override the default).
- **condition** – Any extra constraints that should be observed when determining address satisfiability

**Returns**

A list of concrete addresses.

**concretize_read_addr**(*addr*, *strategies=None*, *condition=None*)

Concretizes an address meant for reading.

**Parameters**

- **addr** – An expression for the address.
- **strategies** – A list of concretization strategies (to override the default).

**Returns**

A list of concrete addresses.

**load**(*addr*, *size=None*, ***, *condition=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

**permissions**(*addr*, *permissions=None*, ***kwargs*)

**map_region**(*addr*, *length*, *permissions*, ***kwargs*)

**unmap_region**(*addr*, *length*, ***kwargs*)

### angr.storage.memory_mixins.bvv_conversion_mixin

**class** angr.storage.memory_mixins.bvv_conversion_mixin.**DataNormalizationMixin**

Bases: *MemoryMixin*

Normalizes the data field for a store and the fallback field for a load to be BVs.

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *fallback=None*, ***kwargs*)

### angr.storage.memory_mixins.conditional_store_mixin

**class** angr.storage.memory_mixins.conditional_store_mixin.**ConditionalMixin**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, ***, *condition=None*, *fallback=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

### angr.storage.memory_mixins.convenient_mappings_mixin

**class** angr.storage.memory_mixins.convenient_mappings_mixin.**ConvenientMappingsMixin**

Bases: *MemoryMixin*

Implements mappings between names and hashes of symbolic variables and these variables themselves.

**__init__**(***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**get_symbolic_addrs**()

**addrs_for_name**(*n*)

Returns addresses that contain expressions that contain a variable named *n*.

**addrs_for_hash**(*h*)

Returns addresses that contain expressions that contain a variable with the hash of *h*.

**replace_all**(*old*, *new*)

Replaces all instances of expression *old* with expression *new*.

**Parameters**

- **old** (BV) – A claripy expression. Must contain at least one named variable (to make it possible to use the name index for speedup).
- **new** (BV) – The new variable to replace it with.

### angr.storage.memory_mixins.default_filler_mixin

**class** angr.storage.memory_mixins.default_filler_mixin.**DefaultFillerMixin**

Bases: *MemoryMixin*

**class** angr.storage.memory_mixins.default_filler_mixin.**SpecialFillerMixin**

Bases: *MemoryMixin*

**__init__**(*special_memory_filler=None*, ***kwargs*)

**class** angr.storage.memory_mixins.default_filler_mixin.**ExplicitFillerMixin**

Bases: *MemoryMixin*

**__init__**(*uninitialized_read_handler=None*, ***kwargs*)

### angr.storage.memory_mixins.dirty_addrs_mixin

**class** angr.storage.memory_mixins.dirty_addrs_mixin.**DirtyAddrsMixin**

Bases: *MemoryMixin*

**store**(*addr*, *data*, *size=None*, ***kwargs*)

### angr.storage.memory_mixins.hex_dumper_mixin

**class** angr.storage.memory_mixins.hex_dumper_mixin.**HexDumperMixin**

Bases: *MemoryMixin*

**hex_dump**(*start*, *size*, *word_size=4*, *words_per_row=4*, *endianness='Iend_BE'*, *symbolic_char='?'*, *unprintable_char='.'*, *solve=False*, *extra_constraints=None*, *inspect=False*, *disable_actions=True*)

Returns a hex dump as a string. The solver, if enabled, is called once for every byte potentially making this function very slow. It is meant to be used mainly as a "visualization" for debugging.

Warning: May read and display more bytes than *size* due to rounding. Particularly, if size is less than, or not a multiple of word_size*words_per_line.

**Parameters**

- **start** – starting address from which to print
- **size** – number of bytes to display
- **word_size** – number of bytes to group together as one space-delimited unit
- **words_per_row** – number of words to display per row of output
- **endianness** – endianness to use when displaying each word (ASCII representation is unchanged)
- **symbolic_char** – the character to display when a byte is symbolic and has multiple solutions

- **unprintable_char** – the character to display when a byte is not printable
- **solve** – whether or not to attempt to solve (warning: can be very slow)
- **extra_constraints** – extra constraints to pass to the solver is solve is True
- **inspect** – whether or not to trigger SimInspect breakpoints for the memory load
- **disable_actions** – whether or not to disable SimActions for the memory load

**Returns**
hex dump as a string

### angr.storage.memory_mixins.inspect_mixin

**class** angr.storage.memory_mixins.inspect_mixin.**InspectMixin**

Bases: *MemoryMixin*

This mixin adds support for the inspect plugin to the memory model.

Inspect breakpoints are called for the following events:

- mem_read (before and after)
- mem_write (before and after)
- reg_read (before and after)
- reg_write (before and after)
- address_concretization_add_constraints (before)

**store**(*addr*, *data*, *size=None*, ***, *condition=None*, *endness=None*, *inspect=True*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *condition=None*, *endness=None*, *inspect=True*, ***kwargs*)

### angr.storage.memory_mixins.javavm_memory_mixin

**class** angr.storage.memory_mixins.javavm_memory_mixin.**JavaVmMemoryMixin**

Bases: *MemoryMixin*

A memory mixin for JavaVM memory.

**__init__**(*memory_id='mem'*, *stack=None*, *heap=None*, *vm_static_table=None*, *load_strategies=None*, *store_strategies=None*, *max_array_size=1000*, ***kwargs*)

**static get_new_uuid**()

Generate a unique id within the scope of the JavaVM memory. This, for example, is used for distinguishing memory objects of the same type (e.g. multiple instances of the same class).

**store**(*addr*, *data*, *frame=0*)

**load**(*addr*, *frame=0*, *none_if_missing=False*)

**push_stack_frame**()

**pop_stack_frame**()

**property stack**

**store_array_element**(*array*, *idx*, *value*)

**store_array_elements**(*array*, *start_idx*, *data*)

Stores either a single element or a range of elements in the array.

**Parameters**

- **array** – Reference to the array.
- **start_idx** – Starting index for the store.
- **data** – Either a single value or a list of values.

**load_array_element**(*array*, *idx*)

**load_array_elements**(*array*, *start_idx*, *no_of_elements*)

Loads either a single element or a range of elements from the array.

**Parameters**

- **array** – Reference to the array.
- **start_idx** – Starting index for the load.
- **no_of_elements** – Number of elements to load.

**concretize_store_idx**(*idx*, *strategies=None*)

Concretizes a store index.

**Parameters**

- **idx** – An expression for the index.
- **strategies** – A list of concretization strategies (to override the default).
- **min_idx** – Minimum value for a concretized index (inclusive).
- **max_idx** – Maximum value for a concretized index (exclusive).

**Returns**

A list of concrete indexes.

**concretize_load_idx**(*idx*, *strategies=None*)

Concretizes a load index.

**Parameters**

- **idx** – An expression for the index.
- **strategies** – A list of concretization strategies (to override the default).
- **min_idx** – Minimum value for a concretized index (inclusive).
- **max_idx** – Maximum value for a concretized index (exclusive).

**Returns**

A list of concrete indexes.

### angr.storage.memory_mixins.keyvalue_memory_mixin

**class** angr.storage.memory_mixins.keyvalue_memory_mixin.**TypedVariable**

Bases: object

TypedVariable is a simple class that holds a value and a type.

**__init__**(*type_*, *value*)

**type**

**value**

**class** angr.storage.memory_mixins.keyvalue_memory_mixin.**KeyValueMemoryMixin**

Bases: *MemoryMixin*

KeyValueMemoryMixin is a mixin that provides a simple key-value store for memory.

**__init__**(**args*, ***kwargs*)

**load**(*addr*, *size=None*, *none_if_missing=False*, ***kwargs*)

**store**(*addr*, *data*, *type_=None*, ***kwargs*)

### angr.storage.memory_mixins.label_merger_mixin

**class** angr.storage.memory_mixins.label_merger_mixin.**LabelMergerMixin**

Bases: *MemoryMixin*

A memory mixin for merging labels. Labels come from SimLabeledMemoryObject.

**__init__**(**args*, ***kwargs*)

### angr.storage.memory_mixins.memory_mixin

**class** angr.storage.memory_mixins.memory_mixin.**MemoryMixin**

Bases: *SimStatePlugin*, Generic

MemoryMixin is the base class for the memory model in angr. It provides a set of methods that should be implemented by memory models. This is done using mixins, where each mixin handles some specific feature of the memory model, only overriding methods that it needs to implement its function. The memory model class itself then combines a set of mixins using inheritence to form the final memory model class.

**SUPPORTS_CONCRETE_LOAD: bool = False**

**__init__**(*memory_id=None*, *endness='Iend_BE'*)

**Parameters**

- **memory_id** (*str | None*)
- **endness** (*str*)

**property category: str**

reg, mem, or file.

**Type**

Return the category of this SimMemory instance. It can be one of the three following categories

**property variable_key_prefix: tuple[Any, ...]**

**find**(*addr*, *data*, *max_search*, ***kwargs*)

**Return type**

tuple[TypeVar(Addr), list[Bool], list[int]]

**Parameters**

- **addr** (*Addr*)

- **data** (*InData*)
- **max_search** (*int*)

**load**(*addr*, *size=None*, ***kwargs*)

**Return type**
TypeVar(OutData)

**Parameters**

- **addr** (*Addr*)
- **size** (*InData | None*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**Return type**
None

**Parameters**

- **addr** (*Addr*)
- **data** (*InData*)
- **size** (*InData | None*)

**compare**(*other*)

**Return type**
bool

**Parameters**
**other** (*Self*)

**permissions**(*addr*, *permissions=None*, ***kwargs*)

**Return type**
BV

**Parameters**

- **addr** (*Addr*)
- **permissions** (*int | claripy.ast.BV | None*)

**map_region**(*addr*, *length*, *permissions*, ***, *init_zero=False*, ***kwargs*)

**Parameters**

- **addr** (*Addr*)
- **length** (*int*)
- **permissions** (*int | claripy.ast.BV*)
- **init_zero** (*bool*)

**unmap_region**(*addr*, *length*, ***kwargs*)

**Parameters**

- **addr** (*Addr*)
- **length** (*int*)

**concrete_load**(*addr*, *size*, *writing=False*, ***kwargs*)

Set SUPPORTS_CONCRETE_LOAD to True and implement concrete_load if reading concrete bytes is faster in this memory model.

**Parameters**

- **addr** – The address to load from.
- **size** – Size of the memory read.
- **writing**

**Return type**
Any

**Returns**
A memoryview into the loaded bytes.

**concrete_run_length**(*addr*, *size*, ***kwargs*)

Return the number of concrete bytes starting at `addr`, capped at `size`.

**Return type**
int

**erase**(*addr*, *size=None*, ***kwargs*)

Set [addr:addr+size) to uninitialized. In many cases this will be faster than overwriting those locations with new values. This is commonly used during static data flow analysis.

**Parameters**

- **addr** (TypeVar(Addr)) – The address to start erasing.
- **size** (int | None) – The number of bytes for erasing.

**Return type**
None

**Returns**
None

**replace_all**(*old*, *new*)

**Parameters**

- **old** (*BV*)
- **new** (*BV*)

**copy_contents**(*dst*, *src*, *size*, *condition=None*, ***kwargs*)

Override this method to provide faster copying of large chunks of data.

**Parameters**

- **dst** (TypeVar(Addr)) – The destination of copying.
- **src** (TypeVar(Addr)) – The source of copying.
- **size** (TypeVar(InData)) – The size of copying.
- **condition** (Bool | None) – The storing condition.
- **kwargs** – Other parameters.

**Returns**
None

### angr.storage.memory_mixins.multi_value_merger_mixin

**class** angr.storage.memory_mixins.multi_value_merger_mixin.**MultiValueMergerMixin**

Bases: *MemoryMixin*

**__init__**(**args*, *element_limit=5*, *annotation_limit=256*, *top_func=None*, *is_top_func=None*, *phi_maker=None*, *merge_into_top=True*, ***kwargs*)

### angr.storage.memory_mixins.name_resolution_mixin

**class** angr.storage.memory_mixins.name_resolution_mixin.**NameResolutionMixin**

Bases: *MemoryMixin*

This mixin allows you to provide register names as load addresses, and will automatically translate this to an offset and size.

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***kwargs*)

### angr.storage.memory_mixins.paged_memory

### Submodules

| |
|---|
| *page_backer_mixins* |
| *paged_memory_mixin* |
| *paged_memory_multivalue_mixin* |
| *pages* |
| *privileged_mixin* |
| *stack_allocation_mixin* |

### angr.storage.memory_mixins.paged_memory.page_backer_mixins

**class** angr.storage.memory_mixins.paged_memory.page_backer_mixins.**NotMemoryview**

Bases: object

**__init__**(*obj*, *offset*, *size*)

**class** angr.storage.memory_mixins.paged_memory.page_backer_mixins.**ClemoryBackerMixin**

Bases: *PagedMemoryMixin*

**__init__**(*cle_memory_backer=None*, ***kwargs*)

**Parameters**
**cle_memory_backer** (*Loader | Clemory | None*)

**class** angr.storage.memory_mixins.paged_memory.page_backer_mixins.**DictBackerMixin**

Bases: *PagedMemoryMixin*

**__init__**(*dict_memory_backer=None*, ***kwargs*)

### angr.storage.memory_mixins.paged_memory.paged_memory_mixin

class angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**PagedMemoryMixin**

Bases: *MemoryMixin*[int | BV | *SimActionObject*, BV, int | BV | *SimActionObject*], Generic

A bottom-level storage mechanism. Dispatches reads to individual pages, the type of which is the PAGE_TYPE class variable.

**SUPPORTS_CONCRETE_LOAD: bool = True**

**PAGE_TYPE: type[PageType]**

**__init__**(*page_size=4096*, *default_permissions=3*, *permissions_map=None*, *page_kwargs=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *endness=None*, ***kwargs*)

**Parameters**

- **addr** (*int*)
- **size** (*int | None*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

**Parameters**

- **addr** (*int*)
- **size** (*int | None*)

**compare**(*other*)

**Return type**
bool

**Parameters**
**other** (PagedMemoryMixin)

**permissions**(*addr*, *permissions=None*, ***kwargs*)

**map_region**(*addr*, *length*, *permissions*, ***, *init_zero=False*, ***kwargs*)

**unmap_region**(*addr*, *length*, ***kwargs*)

**changed_bytes**(*other*)

**Return type**
set[int]

**changed_pages**(*other*)

**Return type**
dict[int, set[int] | None]

**flush_pages**(*white_list*)

Flush all pages not included in the *white_list* by removing their pages. Note, this will not wipe them from memory if they were backed by a memory_backer, it will simply reset them to their initial state. Returns the list of pages that were cleared consisting of *(addr, length)* tuples. :type white_list: :param white_list: white list of regions in the form of (start, end) to exclude from the flush :return: a list of memory page ranges that were flushed :rtype: list

class angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**LabeledPagesMixin**

Bases: *PagedMemoryMixin*

**load_with_labels**(*addr*, *size=None*, *endness=None*, ***kwargs*)

**Return type**
tuple[Base, tuple[tuple[int, int, int, Any]]]

**Parameters**

- **addr** (*int*)
- **size** (*int | None*)

**class** angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**ListPagesMixin**

Bases: *PagedMemoryMixin*

**PAGE_TYPE**

alias of *ListPage*

**class** angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**MVListPagesMixin**

Bases: *PagedMemoryMixin*

**PAGE_TYPE**

alias of *MVListPage*

**__init__**(**args*, *skip_missing_values_during_merging=False*, ***kwargs*)

**class** angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**ListPagesWithLabelsMixin**

Bases: *LabeledPagesMixin*, *ListPagesMixin*

**class** angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**MVListPagesWithLabelsMixin**

Bases: *LabeledPagesMixin*, *MVListPagesMixin*

**class** angr.storage.memory_mixins.paged_memory.paged_memory_mixin.**UltraPagesMixin**

Bases: *PagedMemoryMixin*

**PAGE_TYPE**

alias of *UltraPage*

### angr.storage.memory_mixins.paged_memory.paged_memory_multivalue_mixin

**class** angr.storage.memory_mixins.paged_memory.paged_memory_multivalue_mixin.**PagedMemoryMultiValueMixin**

Bases: *MemoryMixin*

Implement optimizations and fast accessors for the MultiValues-variant of Paged Memory.

**load_annotations**(*addr*, *size*, ***kwargs*)

**Parameters**

- **addr** (*int*)
- **size** (*int*)

### angr.storage.memory_mixins.paged_memory.pages

class angr.storage.memory_mixins.paged_memory.pages.**CooperationBase**

Bases: Generic

Any given subclass of this class which is not a subclass of MemoryMixin should have the property that any subclass it which *is* a subclass of MemoryMixin should all work with the same datatypes

class angr.storage.memory_mixins.paged_memory.pages.**HistoryTrackingMixin**

Bases: *RefcountMixin*, *MemoryMixin*

Tracks the history of memory writes.

**__init__**(**args*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**parents**()

**changed_bytes**(*other*, ***kwargs*)

**Return type**
set[int] | None

**all_bytes_changed_in_history**()

**Return type**
SegmentList

class angr.storage.memory_mixins.paged_memory.pages.**ISPOMixin**

Bases: *MemoryMixin*

An implementation of the International Stateless Persons Organisation, a mixin which should be applied as a bottom layer for memories which have no state and must redirect certain operations to a parent memory. Main usecase is for memory region classes which are stored within other memories, such as pages.

class angr.storage.memory_mixins.paged_memory.pages.**ListPage**

Bases: *MemoryObjectMixin*, *PageBase*

This class implements a page memory mixin with lists as the main content store.

**__init__**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

**load**(*addr*, *size=None*, *endness=None*, *page_addr=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**changed_bytes**(*other*, *page_addr=None*)

**Parameters**

- **other** (ListPage)
- **page_addr** (*int | None*)

class angr.storage.memory_mixins.paged_memory.pages.**MVListPage**

Bases: *MemoryObjectSetMixin*, *PageBase*

MVListPage allows storing multiple values at the same location.

Each store() may take a value or multiple values. Each load() returns an iterator of all values stored at that location.

**__init__**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

**load**(*addr*, *size=None*, *endness=None*, *page_addr=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**Return type**
list[tuple[int, *SimMemoryObject* | *SimLabeledMemoryObject*]]

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**compare**(*other*, *page_addr=None*, *memory=None*, *changed_offsets=None*)

**Return type**
bool

**Parameters**

- **other** (MVListPage)
- **page_addr** (*int | None*)

**changed_bytes**(*other*, *page_addr=None*)

**Parameters**

- **other** (MVListPage)
- **page_addr** (*int | None*)

**content_gen**(*index*)

**class** angr.storage.memory_mixins.paged_memory.pages.**MemoryObjectMixin**

Bases: *CooperationBase*[*SimMemoryObject*]

Uses SimMemoryObjects in region storage. With this, load will return a list of tuple (address, MO) and store will take a MO.

**class** angr.storage.memory_mixins.paged_memory.pages.**PageBase**

Bases: *HistoryTrackingMixin*, *RefcountMixin*, *CooperationBase*, *ISPOMixin*, *PermissionsMixin*, *MemoryMixin*

This is a fairly succinct definition of the contract between PagedMemoryMixin and its constituent pages:

- Pages must implement the MemoryMixin model for loads, stores, copying, merging, etc
- However, loading/storing may not necessarily use the same data domain as PagedMemoryMixin. In order to do more efficient loads/stores across pages, we use the CooperationBase interface which allows the page class to determine how to generate and unwrap the objects which are actually stored.
- To support COW, we use the RefcountMixin and the ISPOMixin (which adds the contract element that `memory=self` be passed to every method call)
- Pages have permissions associated with them, stored in the PermissionsMixin.

Read the docstrings for each of the constituent classes to understand the nuances of their functionalities

**class** angr.storage.memory_mixins.paged_memory.pages.**PermissionsMixin**

Bases: *MemoryMixin*

This mixin adds a permissions_bits field and properties for extracting the read/write/exec permissions. It does NOT add permissions checking.

**__init__**(*permissions=None*, ***kwargs*)

**Parameters**
**permissions** (*int | BV | None*)

property perm_read

property perm_write

property perm_exec

class angr.storage.memory_mixins.paged_memory.pages.RefcountMixin

Bases: *MemoryMixin*

This mixin adds a locked reference counter and methods to manipulate it, to facilitate copy-on-write optimizations.

__init__(***kwargs*)

acquire_unique()

Call this function to return a version of this page which can be used for writing, which may or may not be the same object as before. If you use this you must immediately replace the shared reference you previously had with the new unique copy.

acquire_shared()

Call this function to indicate that this page has had a reference added to it and must be copied before it can be acquired uniquely again. Creating the object implicitly starts it with one shared reference.

**Return type**
None

release_shared()

Call this function to indicate that this page has had a shared reference to it released

**Return type**
None

class angr.storage.memory_mixins.paged_memory.pages.UltraPage

Bases: *MemoryObjectMixin*, *PageBase*

Default page implementation

SUPPORTS_CONCRETE_LOAD: bool = True

__init__(*memory=None*, *init_zero=False*, ***kwargs*)

classmethod new_from_shared(*data*, *memory=None*, ***kwargs*)

load(*addr*, *size=None*, *page_addr=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

store(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *page_addr=None*, *cooperate=False*, ***kwargs*)

**Parameters**

- **data** (*int | SimMemoryObject*)
- **size** (*int | None*)

concrete_run_length(*addr*, *size*, ***kwargs*)

Return the number of concrete bytes at addr, capped at size.

**Return type**
int

**changed_bytes**(*other*, *page_addr=None*)

**Return type**
set[int]

**replace_all_with_offsets**(*offsets*, *old*, *new*, *memory=None*)

**Parameters**

- **offsets** (*Iterable[int]*)
- **old** (*BV*)
- **new** (*BV*)

## Submodules


| |
|---|
| *base* |
| *cooperation* |
| *history_tracking_mixin* |
| *ispo_mixin* |
| *list_page* |
| *multi_values* |
| *mv_list_page* |
| *permissions_mixin* |
| *refcount_mixin* |
| *symbolic_bitmap* |
| *ultra_page* |


### angr.storage.memory_mixins.paged_memory.pages.base

**class** angr.storage.memory_mixins.paged_memory.pages.base.**PageBase**

Bases: *HistoryTrackingMixin*, *RefcountMixin*, *CooperationBase*, *ISPOMixin*, *PermissionsMixin*, *MemoryMixin*

This is a fairly succinct definition of the contract between PagedMemoryMixin and its constituent pages:

- Pages must implement the MemoryMixin model for loads, stores, copying, merging, etc
- However, loading/storing may not necessarily use the same data domain as PagedMemoryMixin. In order to do more efficient loads/stores across pages, we use the CooperationBase interface which allows the page class to determine how to generate and unwrap the objects which are actually stored.
- To support COW, we use the RefcountMixin and the ISPOMixin (which adds the contract element that `memory=self` be passed to every method call)
- Pages have permissions associated with them, stored in the PermissionsMixin.

Read the docstrings for each of the constituent classes to understand the nuances of their functionalities

### angr.storage.memory_mixins.paged_memory.pages.cooperation

**class** angr.storage.memory_mixins.paged_memory.pages.cooperation.**CooperationBase**

Bases: Generic

Any given subclass of this class which is not a subclass of MemoryMixin should have the property that any subclass it which *is* a subclass of MemoryMixin should all work with the same datatypes

class angr.storage.memory_mixins.paged_memory.pages.cooperation.**MemoryObjectMixin**

Bases: *CooperationBase*[*SimMemoryObject*]

Uses SimMemoryObjects in region storage. With this, load will return a list of tuple (address, MO) and store will take a MO.

class angr.storage.memory_mixins.paged_memory.pages.cooperation.**MemoryObjectSetMixin**

Bases: *CooperationBase*

Uses sets of SimMemoryObjects in region storage.

class angr.storage.memory_mixins.paged_memory.pages.cooperation.**BasicClaripyCooperation**

Bases: *CooperationBase*

Mix this (along with PageBase) into a storage class which supports loading and storing claripy bitvectors and it will be able to work as a page in the paged memory model.

### angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin

class angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.**HistoryTrackingMixin**

Bases: *RefcountMixin*, *MemoryMixin*

Tracks the history of memory writes.

**__init__**(**args*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**parents**()

**changed_bytes**(*other*, ***kwargs*)

**Return type**
set[int] | None

**all_bytes_changed_in_history**()

**Return type**
SegmentList

### angr.storage.memory_mixins.paged_memory.pages.ispo_mixin

exception angr.storage.memory_mixins.paged_memory.pages.ispo_mixin.**MissingMemoryError**

Bases: Exception

Raised when a memory kwarg is not passed to a stateless object.

class angr.storage.memory_mixins.paged_memory.pages.ispo_mixin.**ISPOMixin**

Bases: *MemoryMixin*

An implementation of the International Stateless Persons Organisation, a mixin which should be applied as a bottom layer for memories which have no state and must redirect certain operations to a parent memory. Main usecase is for memory region classes which are stored within other memories, such as pages.

### angr.storage.memory_mixins.paged_memory.pages.list_page

**class** angr.storage.memory_mixins.paged_memory.pages.list_page.**ListPage**

Bases: *MemoryObjectMixin*, *PageBase*

This class implements a page memory mixin with lists as the main content store.

**__init__**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

**load**(*addr*, *size=None*, *endness=None*, *page_addr=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**changed_bytes**(*other*, *page_addr=None*)

**Parameters**

- **other** (ListPage)
- **page_addr** (*int | None*)

### angr.storage.memory_mixins.paged_memory.pages.multi_values

**class** angr.storage.memory_mixins.paged_memory.pages.multi_values.**MultiValues**

Bases: Generic

Represents a byte vector where each byte can have one or multiple values.

As an implementation optimization (so that we do not create excessive sets and dicts), self._single_value stores a claripy AST when this MultiValues object represents only one value at offset 0.

**__init__**(*v=None*, *offset_to_values=None*)

**Parameters**

- **v** (*MVType | MultiValues[MVType] | dict[int, set[MVType]] | None*)
- **offset_to_values** (*dict[int, set[MVType]] | None*)

**add_value**(*offset*, *value*)

**Return type**
None

**Parameters**

- **offset** (*int*)
- **value** (*MVType*)

**one_value**(*strip_annotations=False*)

**Return type**
Optional[TypeVar(MVType, bound= BV | FP)]

**Parameters**
**strip_annotations** (*bool*)

**merge**(*mv*)

**Return type**
*MultiValues*[TypeVar(MVType, bound= BV | FP)]

**Parameters**
**mv** (MultiValues[*MVType*])

**keys**()

**Return type**
set[int]

**values**()

**Return type**
Iterator[set[TypeVar(MVType, bound= BV | FP)]]

**items**()

**Return type**
Iterator[tuple[int, set[TypeVar(MVType, bound= BV | FP)]]]

**count**()

**Return type**
int

**extract**(*offset*, *length*, *endness*)

**Return type**
*MultiValues*[BV]

**Parameters**

- **self** (MultiValues[*BV*])
- **offset** (*int*)
- **length** (*int*)
- **endness** (*str*)

**concat**(*other*)

**Return type**
*MultiValues*[BV]

**Parameters**

- **self** (MultiValues[*BV*])
- **other** (MultiValues[*BV*] | *BV* | bytes)

angr.storage.memory_mixins.paged_memory.pages.multi_values.**mv_is_bv**(*mv*)

**Return type**
TypeGuard[*MultiValues*[BV]]

**Parameters**
**mv** (MultiValues[*Any*])

angr.storage.memory_mixins.paged_memory.pages.multi_values.**mv_is_fp**(*mv*)

**Return type**
TypeGuard[*MultiValues*[FP]]

**Parameters**
**mv** (MultiValues[*Any*])

### angr.storage.memory_mixins.paged_memory.pages.mv_list_page

**class** angr.storage.memory_mixins.paged_memory.pages.mv_list_page.**MVListPage**

Bases: *MemoryObjectSetMixin*, *PageBase*

MVListPage allows storing multiple values at the same location.

Each store() may take a value or multiple values. Each load() returns an iterator of all values stored at that location.

**__init__**(*memory=None*, *content=None*, *sinkhole=None*, *mo_cmp=None*, ***kwargs*)

**load**(*addr*, *size=None*, *endness=None*, *page_addr=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**Return type**
list[tuple[int, *SimMemoryObject* | *SimLabeledMemoryObject*]]

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**compare**(*other*, *page_addr=None*, *memory=None*, *changed_offsets=None*)

**Return type**
bool

**Parameters**

- **other** (MVListPage)
- **page_addr** (*int | None*)

**changed_bytes**(*other*, *page_addr=None*)

**Parameters**

- **other** (MVListPage)
- **page_addr** (*int | None*)

**content_gen**(*index*)

### angr.storage.memory_mixins.paged_memory.pages.permissions_mixin

**class** angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.**PermissionsMixin**

Bases: *MemoryMixin*

This mixin adds a permissions_bits field and properties for extracting the read/write/exec permissions. It does NOT add permissions checking.

**__init__**(*permissions=None*, ***kwargs*)

**Parameters**
**permissions** (*int | BV | None*)

**property perm_read**

**property perm_write**

**property perm_exec**

### angr.storage.memory_mixins.paged_memory.pages.refcount_mixin

class angr.storage.memory_mixins.paged_memory.pages.refcount_mixin.**RefcountMixin**

Bases: *MemoryMixin*

This mixin adds a locked reference counter and methods to manipulate it, to facilitate copy-on-write optimizations.

**__init__**(***kwargs*)

**acquire_unique**()

Call this function to return a version of this page which can be used for writing, which may or may not be the same object as before. If you use this you must immediately replace the shared reference you previously had with the new unique copy.

**acquire_shared**()

Call this function to indicate that this page has had a reference added to it and must be copied before it can be acquired uniquely again. Creating the object implicitly starts it with one shared reference.

**Return type**
None

**release_shared**()

Call this function to indicate that this page has had a shared reference to it released

**Return type**
None

### angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap

class angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap.**SymbolicBitmap**

Bases: object

Tracks, for every byte of an UltraPage, whether that byte is symbolic or not.

The map is stored as a bitmap where the i-th bit of the map lives in bit `i & 7` of byte `i >> 3` (least-significant bit first). This makes `int.from_bytes(..., "little")` a direct view of the map as a big integer and lets range scans be done with `bit_length()` instead of a Python-level loop.

A page whose map is uniform (all bytes are symbolic or concrete) does not store any map.

All ranges are half-open `[start, stop)` and are assumed to lie within `[0, size]`.

**__init__**(*size*, *value=0*)

**Parameters**

- **size** (*int*)
- **value** (*int*)

**size**

**property nbytes: int**

The number of bytes of backing store this map currently occupies (0 while uniform).

**property uniform_value: int | None**

Return 0/1 if every byte of the page has that symbolic-ness, or None if the map is mixed.

`get`(*i*)

Return 1 if byte `i` is symbolic, 0 otherwise.

**Return type**
`int`

**Parameters**
**i** (*int*)

`set`(*i*, *value*)

Mark byte `i` as symbolic (`value` truthy) or concrete.

**Return type**
`None`

**Parameters**

- **i** (*int*)
- **value** (*int*)

`set_range`(*start*, *stop*)

Mark `[start, stop)` as symbolic.

**Return type**
`None`

**Parameters**

- **start** (*int*)
- **stop** (*int*)

`clear_range`(*start*, *stop*)

Mark `[start, stop)` as concrete.

**Return type**
`None`

**Parameters**

- **start** (*int*)
- **stop** (*int*)

`next_set`(*start*, *stop*)

Return the smallest `i` in `[start, stop)` whose byte is symbolic, or `stop` if there is none.

A whole range is turned into one big integer and located with `(w & -w).bit_length()` rather than being walked bit by bit. Long ranges are probed 64 bits at a time first.

**Return type**
`int`

**Parameters**

- **start** (*int*)
- **stop** (*int*)

`next_clear`(*start*, *stop*)

Return the smallest `i` in `[start, stop)` whose byte is concrete, or `stop` if there is none.

**Return type**
`int`

**Parameters**

- **start** (*int*)
- **stop** (*int*)

**all_set**(*start*, *stop*)

Return True if every byte in `[start, stop)` is symbolic.

**Return type**
bool

**Parameters**

- **start** (*int*)
- **stop** (*int*)

**any_set**(*start*, *stop*)

Return True if any byte in `[start, stop)` is symbolic.

**Return type**
bool

**Parameters**

- **start** (*int*)
- **stop** (*int*)

**view**(*start*, *stop*)

Return the packed bits covering `[start, stop)`: bit `i` of the result is byte `start + i` of the page.

When `start` is byte-aligned, the result is a writable view of this map's own backing store, so a native consumer (e.g., unicorn engine) can update the page's symbolic-ness in place. Returning such a view pins the backing store: it may no longer be dropped in favor of the uniform representation.

An unaligned `start` cannot be expressed as a view of the backing store, so the bits are shifted into place and returned read-only. Consumers like unicorn engine should not use this view for writing!

**Return type**
memoryview

**Parameters**

- **start** (*int*)
- **stop** (*int*)

**copy**()

**Return type**
*SymbolicBitmap*

### angr.storage.memory_mixins.paged_memory.pages.ultra_page

**class** angr.storage.memory_mixins.paged_memory.pages.ultra_page.**UltraPage**

Bases: *MemoryObjectMixin*, *PageBase*

Default page implementation

**SUPPORTS_CONCRETE_LOAD: bool = True**

**__init__**(*memory=None*, *init_zero=False*, ***kwargs*)

**classmethod new_from_shared**(*data*, *memory=None*, ***kwargs*)

**load**(*addr*, *size=None*, *page_addr=None*, *endness=None*, *memory=None*, *cooperate=False*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, *endness=None*, *memory=None*, *page_addr=None*, *cooperate=False*, ***kwargs*)

**Parameters**

- **data** (*int | SimMemoryObject*)
- **size** (*int | None*)

**concrete_run_length**(*addr*, *size*, ***kwargs*)

Return the number of concrete bytes at `addr`, capped at `size`.

**Return type**
int

**changed_bytes**(*other*, *page_addr=None*)

**Return type**
set[int]

**replace_all_with_offsets**(*offsets*, *old*, *new*, *memory=None*)

**Parameters**

- **offsets** (*Iterable[int]*)
- **old** (*BV*)
- **new** (*BV*)

### angr.storage.memory_mixins.paged_memory.privileged_mixin

**class** angr.storage.memory_mixins.paged_memory.privileged_mixin.**PrivilegedPagingMixin**

Bases: *PagedMemoryMixin*

A mixin for paged memory models which will raise SimSegfaultExceptions if STRICT_PAGE_ACCESS is enabled and a segfault condition is detected.

Segfault conditions include: - getting a page for reading which is non-readable - getting a page for writing which is non-writable - creating a page

The latter condition means that this should be inserted under any mixins which provide other implementations of `_initialize_page`.

### angr.storage.memory_mixins.paged_memory.stack_allocation_mixin

**class** angr.storage.memory_mixins.paged_memory.stack_allocation_mixin.**StackAllocationMixin**

Bases: *PagedMemoryMixin*

This mixin adds automatic allocation for a stack region based on the stack_end and stack_size parameters.

**__init__**(*stack_end=None*, *stack_size=None*, *stack_perms=None*, ***kwargs*)

**allocate_stack_pages**(*addr*, *size*, ***kwargs*)

Pre-allocates pages for the stack without triggering any logic related to reading from them.

**Parameters**

- **addr** (int) – The highest address that should be mapped

- **size** (int) – The number of bytes to be allocated. byte 1 is the one at addr, byte 2 is the one before that, and so on.

**Returns**
A list of the new page objects

### angr.storage.memory_mixins.regioned_memory

**class** angr.storage.memory_mixins.regioned_memory.**AbstractMergerMixin**

Bases: *MemoryMixin*

AbstractMergerMixin handles merging initialized values.

**class** angr.storage.memory_mixins.regioned_memory.**MemoryRegionMetaMixin**

Bases: *MemoryMixin*

**__init__**(*related_function_addr=None*, ***kwargs*)

**property is_stack**

**property related_function_addr**

**get_abstract_locations**(*addr*, *size*)

Get a list of abstract locations that is within the range of [addr, addr + size]

This implementation is pretty slow. But since this method won't be called frequently, we can live with the bad implementation for now.

**Parameters**

- **addr** – Starting address of the memory region.
- **size** – Size of the memory region, in bytes.

**Returns**
A list of covered AbstractLocation objects, or an empty list if there is none.

**store**(*addr*, *data*, *size=None*, ***, *bbl_addr=None*, *stmt_id=None*, *ins_addr=None*, *endness=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *bbl_addr=None*, *stmt_idx=None*, *ins_addr=None*, ***kwargs*)

**dbg_print**(*indent=0*)

Print out debugging information

**class** angr.storage.memory_mixins.regioned_memory.**RegionCategoryMixin**

Bases: *MemoryMixin*

**property category**

reg, mem, or file.

**Type**
Return the category of this SimMemory instance. It can be one of the three following categories

**class** angr.storage.memory_mixins.regioned_memory.**RegionedAddressConcretizationMixin**

Bases: *MemoryMixin*

**__init__**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

**class** angr.storage.memory_mixins.regioned_memory.**RegionedMemoryMixin**

Bases: *MemoryMixin*

Regioned memory. This mixin manages multiple memory regions. Each address is represented as a tuple of (region ID, offset into the region), which is called a regioned address.

Converting absolute addresses into regioned addresses: We map an absolute address to a region by looking up which region this address belongs to in the region map. Currently this is only enabled for stack. Heap support has not landed yet.

When start analyzing a function, the user should call set_stack_address_mapping() to create a new region mapping. Likewise, when exiting from a function, the user should cancel the previous mapping by calling unset_stack_address_mapping().

**__init__**(*write_targets_limit=2048*, *read_targets_limit=4096*, *stack_region_map=None*, *generic_region_map=None*, *stack_size=65536*, *cle_memory_backer=None*, *dict_memory_backer=None*, *regioned_memory_cls=None*, ***kwargs*)

**Parameters**

- **write_targets_limit** (*int*)
- **read_targets_limit** (*int*)
- **stack_region_map** (RegionMap | *None*)
- **generic_region_map** (RegionMap | *None*)
- **stack_size** (*int*)
- **cle_memory_backer** (*None*)
- **dict_memory_backer** (*dict* | *None*)
- **regioned_memory_cls** (type | *None*)

**load**(*addr*, *size=None*, ***, *endness=None*, *condition=None*, ***kwargs*)

**Parameters**

- **size** (*BV* | *int* | *None*)
- **condition** (*Bool* | *None*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

**Parameters**
**size** (*int* | *None*)

**find**(*addr*, *data*, *max_search*, ***kwargs*)

**Parameters**
**addr** (*int* | *Bits*)

**replace_all**(*old*, *new*)

**Parameters**

- **old** (*BV*)
- **new** (*BV*)

**set_stack_address_mapping**(*absolute_address*, *region_id*, *related_function_address=None*)

Create a new mapping between an absolute address (which is the base address of a specific stack frame) and a region ID.

**Parameters**

- **absolute_address** (int) – The absolute memory address.
- **region_id** (str) – The region ID.
- **related_function_address** (int | None) – Related function address.

**unset_stack_address_mapping**(*absolute_address*)

Remove a stack mapping.

**Parameters**
**absolute_address** (int) – An absolute memory address that is the base address of the stack frame to destroy.

**stack_id**(*function_address*)

Return a memory region ID for a function. If the default region ID exists in the region mapping, an integer will appended to the region name. In this way we can handle recursive function calls, or a function that appears more than once in the call frame.

This also means that *stack_id()* should only be called when creating a new stack frame for a function. You are not supposed to call this function every time you want to map a function address to a stack ID.

**Parameters**
**function_address** (int) – Address of the function.

**Return type**
str

**Returns**
ID of the new memory region.

**set_stack_size**(*size*)

**Parameters**
**size** (*int*)

**class** angr.storage.memory_mixins.regioned_memory.**StaticFindMixin**

Bases: *SmartFindMixin*

Implements data finding for abstract memory.

**find**(*addr*, *data*, *max_search*, ***, *default=None*, *endness=None*, *chunk_size=None*, *max_symbolic_bytes=None*, *condition=None*, *char_size=1*, ***kwargs*)

## Submodules

| |
|---|
| *abstract_address_descriptor* |
| *abstract_merger_mixin* |
| *region_category_mixin* |
| *region_data* |
| *region_meta_mixin* |
| *regioned_address_concretization_mixin* |
| *regioned_memory_mixin* |

continues on next page

Table 114 – continued from previous page

| |
|---|
| *static_find_mixin* |

### angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor

**class** angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor.**AbstractAddressDescriptor**

Bases: object

AbstractAddressDescriptor describes a list of region+offset tuples. It provides a convenient way for accessing the cardinality (the total number of addresses) without enumerating or creating all addresses in static mode.

**__init__**()

**property cardinality**

**add_regioned_address**(*region*, *addr*)

**Parameters**

- **region** (*str*)
- **addr** (*BV*)

**clear**()

### angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin

**class** angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin.**AbstractMergerMixin**

Bases: *MemoryMixin*

AbstractMergerMixin handles merging initialized values.

### angr.storage.memory_mixins.regioned_memory.region_category_mixin

**class** angr.storage.memory_mixins.regioned_memory.region_category_mixin.**RegionCategoryMixin**

Bases: *MemoryMixin*

**property category**

reg, mem, or file.

**Type**

Return the category of this SimMemory instance. It can be one of the three following categories

### angr.storage.memory_mixins.regioned_memory.region_data

**class** angr.storage.memory_mixins.regioned_memory.region_data.**AddressWrapper**

Bases: object

AddressWrapper is used in SimAbstractMemory, which provides extra meta information for an address (or a ValueSet object) that is normalized from an integer/BVV/StridedInterval.

__init__(*region*, *region_base_addr*, *address*, *is_on_stack*, *function_address*)

Constructor for the class AddressWrapper.

**Parameters**

- **region** (str) – Name of the memory regions it belongs to.
- **region_base_addr** (int) – Base address of the memory region
- **address** – An address (not a ValueSet object).
- **is_on_stack** (bool) – Whether this address is on a stack region or not.
- **function_address** (int | None) – Related function address (if any).

region

region_base_addr

address

is_on_stack

function_address

to_valueset(*state*)

Convert to a ValueSet instance

**Parameters**
**state** – A state

**Returns**
The converted ValueSet instance

class angr.storage.memory_mixins.regioned_memory.region_data.**RegionDescriptor**

Bases: object

Descriptor for a memory region ID.

__init__(*region_id*, *base_address*, *related_function_address=None*)

region_id

base_address

related_function_address

class angr.storage.memory_mixins.regioned_memory.region_data.**RegionMap**

Bases: object

Mostly used in SimAbstractMemory, RegionMap stores a series of mappings between concrete memory address ranges and memory regions, like stack frames and heap regions.

__init__(*is_stack*)

Constructor

**Parameters**
**is_stack** – Whether this is a region map for stack frames or not. Different strategies apply for stack regions.

property is_empty

**property stack_base**

**property region_ids**

**copy**(*memo*)

**map**(*absolute_address*, *region_id*, *related_function_address=None*)

Add a mapping between an absolute address and a region ID. If this is a stack region map, all stack regions beyond (lower than) this newly added regions will be discarded.

**Parameters**

- **absolute_address** – An absolute memory address.
- **region_id** – ID of the memory region.
- **related_function_address** – A related function address, mostly used for stack regions.

**unmap_by_address**(*absolute_address*)

Removes a mapping based on its absolute address.

**Parameters**
**absolute_address** – An absolute address

**absolutize**(*region_id*, *relative_address*)

Convert a relative address in some memory region to an absolute address.

**Parameters**

- **region_id** – The memory region ID
- **relative_address** – The relative memory offset in that memory region

**Returns**
An absolute address if converted, or an exception is raised when region id does not exist.

**relativize**(*absolute_address*, *target_region_id=None*)

Convert an absolute address to the memory offset in a memory region.

Note that if an address belongs to heap region is passed in to a stack region map, it will be converted to an offset included in the closest stack frame, and vice versa for passing a stack address to a heap region. Therefore you should only pass in address that belongs to the same category (stack or non-stack) of this region map.

**Parameters**
**absolute_address** – An absolute memory address

**Returns**
A tuple of the closest region ID, the relative offset, and the related function address.

### angr.storage.memory_mixins.regioned_memory.region_meta_mixin

**class** angr.storage.memory_mixins.regioned_memory.region_meta_mixin.**Segment**

Bases: object

Segment represents a continuous memory region.

**__init__**(*offset*, *size=0*)

class angr.storage.memory_mixins.regioned_memory.region_meta_mixin.**AbstractLocation**

Bases: object

AbstractLocation represents a location in memory.

**__init__**(*bbl_key*, *stmt_id*, *region_id*, *segment_list=None*, *region_offset=None*, *size=None*)

**property basicblock_key**

**property statement_id**

**property region**

**property segments**

**update**(*region_offset*, *size*)

**copy**()

**merge**(*other*)

class angr.storage.memory_mixins.regioned_memory.region_meta_mixin.**MemoryRegionMetaMixin**

Bases: *MemoryMixin*

**__init__**(*related_function_addr=None*, ***kwargs*)

**property is_stack**

**property related_function_addr**

**get_abstract_locations**(*addr*, *size*)

Get a list of abstract locations that is within the range of [addr, addr + size]

This implementation is pretty slow. But since this method won't be called frequently, we can live with the bad implementation for now.

**Parameters**

- **addr** – Starting address of the memory region.
- **size** – Size of the memory region, in bytes.

**Returns**

A list of covered AbstractLocation objects, or an empty list if there is none.

**store**(*addr*, *data*, *size=None*, ***, *bbl_addr=None*, *stmt_id=None*, *ins_addr=None*, *endness=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *bbl_addr=None*, *stmt_idx=None*, *ins_addr=None*, ***kwargs*)

**dbg_print**(*indent=0*)

Print out debugging information

### angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin

class angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin.**RegionedAddressConcretizationMixin**

Bases: *MemoryMixin*

**__init__**(*read_strategies=None*, *write_strategies=None*, ***kwargs*)

### angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin

**class**
angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.**RegionedMemoryMixin**

Bases: *MemoryMixin*

Regioned memory. This mixin manages multiple memory regions. Each address is represented as a tuple of (region ID, offset into the region), which is called a regioned address.

Converting absolute addresses into regioned addresses: We map an absolute address to a region by looking up which region this address belongs to in the region map. Currently this is only enabled for stack. Heap support has not landed yet.

When start analyzing a function, the user should call set_stack_address_mapping() to create a new region mapping. Likewise, when exiting from a function, the user should cancel the previous mapping by calling unset_stack_address_mapping().

**__init__**(*write_targets_limit=2048*, *read_targets_limit=4096*, *stack_region_map=None*, *generic_region_map=None*, *stack_size=65536*, *cle_memory_backer=None*, *dict_memory_backer=None*, *regioned_memory_cls=None*, ***kwargs*)

**Parameters**

- **write_targets_limit** (*int*)
- **read_targets_limit** (*int*)
- **stack_region_map** (RegionMap | *None*)
- **generic_region_map** (RegionMap | *None*)
- **stack_size** (*int*)
- **cle_memory_backer** (*None*)
- **dict_memory_backer** (*dict* | *None*)
- **regioned_memory_cls** (type | *None*)

**load**(*addr*, *size=None*, ***, *endness=None*, *condition=None*, ***kwargs*)

**Parameters**

- **size** (*BV* | *int* | *None*)
- **condition** (*Bool* | *None*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

**Parameters**
**size** (*int* | *None*)

**find**(*addr*, *data*, *max_search*, ***kwargs*)

**Parameters**
**addr** (*int* | *Bits*)

**replace_all**(*old*, *new*)

**Parameters**

- **old** (*BV*)
- **new** (*BV*)

**set_stack_address_mapping**(*absolute_address*, *region_id*, *related_function_address=None*)

Create a new mapping between an absolute address (which is the base address of a specific stack frame) and a region ID.

**Parameters**

- **absolute_address** (int) – The absolute memory address.
- **region_id** (str) – The region ID.
- **related_function_address** (int | None) – Related function address.

**unset_stack_address_mapping**(*absolute_address*)

Remove a stack mapping.

**Parameters**
**absolute_address** (int) – An absolute memory address that is the base address of the stack frame to destroy.

**stack_id**(*function_address*)

Return a memory region ID for a function. If the default region ID exists in the region mapping, an integer will appended to the region name. In this way we can handle recursive function calls, or a function that appears more than once in the call frame.

This also means that *stack_id()* should only be called when creating a new stack frame for a function. You are not supposed to call this function every time you want to map a function address to a stack ID.

**Parameters**
**function_address** (int) – Address of the function.

**Return type**
str

**Returns**
ID of the new memory region.

**set_stack_size**(*size*)

**Parameters**
**size** (*int*)

### angr.storage.memory_mixins.regioned_memory.static_find_mixin

**class** angr.storage.memory_mixins.regioned_memory.static_find_mixin.**StaticFindMixin**

Bases: *SmartFindMixin*

Implements data finding for abstract memory.

**find**(*addr*, *data*, *max_search*, ***, *default=None*, *endness=None*, *chunk_size=None*, *max_symbolic_bytes=None*, *condition=None*, *char_size=1*, ***kwargs*)

### angr.storage.memory_mixins.simple_interface_mixin

**class** angr.storage.memory_mixins.simple_interface_mixin.**SimpleInterfaceMixin**

Bases: *MemoryMixin*

**load**(*addr*, *size=None*, ***, *endness=None*, *condition=None*, *fallback=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, *condition=None*, ***kwargs*)

### angr.storage.memory_mixins.simplification_mixin

**class** angr.storage.memory_mixins.simplification_mixin.**SimplificationMixin**

Bases: *MemoryMixin*

**store**(*addr*, *data*, *size=None*, ***kwargs*)

### angr.storage.memory_mixins.size_resolution_mixin

**class** angr.storage.memory_mixins.size_resolution_mixin.**SizeNormalizationMixin**

Bases: *MemoryMixin*

Provides basic services related to normalizing sizes. After this mixin, sizes will always be a plain int. Assumes that the data is a BV.

- load will throw a TypeError if no size is provided
- store will default to len(data)//byte_width if no size is provided

**load**(*addr*, *size=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

**class** angr.storage.memory_mixins.size_resolution_mixin.**SizeConcretizationMixin**

Bases: *MemoryMixin*

This mixin allows memory to process symbolic sizes. It will not touch any sizes which are not ASTs with non-BVV ops. Assumes that the data is a BV.

- symbolic load sizes will be concretized as their maximum and a warning will be logged
- symbolic store sizes will be dispatched as several conditional stores with concrete sizes

**__init__**(*concretize_symbolic_write_size=False*, *max_concretize_count=256*, *max_symbolic_size=4194304*, *raise_memory_limit_error=False*, *size_limit=257*, ***kwargs*)

**Parameters**

- **concretize_symbolic_write_size** (*bool*)
- **max_concretize_count** (*int | None*)
- **max_symbolic_size** (*int*)
- **raise_memory_limit_error** (*bool*)
- **size_limit** (*int*)

**load**(*addr*, *size=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *condition=None*, ***kwargs*)

### angr.storage.memory_mixins.slotted_memory

**class** angr.storage.memory_mixins.slotted_memory.**SlottedMemoryMixin**

Bases: *MemoryMixin*

**__init__**(*width=None*, ***kwargs*)

**load**(*addr*, *size=None*, ***, *endness=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***, *endness=None*, ***kwargs*)

**changed_bytes**(*other*)

#### angr.storage.memory_mixins.smart_find_mixin

class angr.storage.memory_mixins.smart_find_mixin.**SmartFindMixin**

Bases: *MemoryMixin*

Memory mixin providing basic searching over concrete and symbolic data.

**find**(*addr*, *data*, *max_search*, *, *default=None*, *endness=None*, *chunk_size=None*, *max_symbolic_bytes=None*, *condition=None*, *char_size=1*, ***kwargs*)

#### angr.storage.memory_mixins.symbolic_merger_mixin

class angr.storage.memory_mixins.symbolic_merger_mixin.**SymbolicMergerMixin**

Bases: *MemoryMixin*

#### angr.storage.memory_mixins.top_merger_mixin

class angr.storage.memory_mixins.top_merger_mixin.**TopMergerMixin**

Bases: *MemoryMixin*

A memory mixin for merging values in memory to TOP.

**__init__**(**args*, *top_func=None*, ***kwargs*)

#### angr.storage.memory_mixins.underconstrained_mixin

class angr.storage.memory_mixins.underconstrained_mixin.**UnderconstrainedMixin**

Bases: *MemoryMixin*

**__init__**(**args*, ***kwargs*)

**load**(*addr*, *size=None*, ***kwargs*)

**store**(*addr*, *data*, *size=None*, ***kwargs*)

#### angr.storage.memory_mixins.unwrapper_mixin

class angr.storage.memory_mixins.unwrapper_mixin.**UnwrapperMixin**

Bases: *MemoryMixin*

This mixin processes SimActionObjects by passing on their .ast field.

**store**(*addr*, *data*, *size=None*, *, *condition=None*, ***kwargs*)

**load**(*addr*, *size=None*, *, *condition=None*, *fallback=None*, ***kwargs*)

**find**(*addr*, *data*, *max_search*, *, *default=None*, ***kwargs*)

### 10.44.3 angr.storage.memory_object

angr.storage.memory_object.**obj_bit_size**(*o*)

class angr.storage.memory_object.**SimMemoryObject**

Bases: object

A SimMemoryObject is a reference to a byte or several bytes in a specific object in memory. It should be used only by the bottom layer of memory.

**__init__**(*obj*, *base*, *endness*, *length=None*, *byte_width=8*)

**is_bytes**

**base**

**object: BV | FP**

**length**

**endness**

**size**()

**property variables**

**property symbolic**

**property last_addr**

**concrete_bytes**(*offset*, *size*)

**Return type**
bytes | None

**Parameters**

- **offset** (*int*)
- **size** (*int*)

**includes**(*x*)

**bytes_at**(*addr*, *length*, *allow_concrete=False*, *endness='Iend_BE'*)

**class** angr.storage.memory_object.**SimLabeledMemoryObject**

Bases: *SimMemoryObject*

SimLabeledMemoryObject is a SimMemoryObject with a label

**__init__**(*obj*, *base*, *endness*, *length=None*, *byte_width=8*, *label=None*)

**label**

angr.storage.memory_object.**bv_slice**(*value*, *offset*, *size*, *rev*, *bw*)

Extremely cute utility to pretend you've serialized a value to stored bytes, sliced it a la python slicing, and then deserialized those bytes to an integer again.

**Parameters**

- **value** (BV) – The bitvector to slice
- **offset** (int) – The byte offset from the first stored byte to slice from, or a negative offset from the end.
- **size** (int) – The number of bytes to return. If None, return all bytes from the offset to the end. If larger than the number of bytes from the offset to the end, return all bytes from the offset to the end.
- **rev** (bool) – Whether the pretend-serialization should be little-endian
- **bw** (int) – The byte width

**Return type**
BV

**Returns**
The new bitvector

## 10.45 angr.tablespecs

**class** angr.tablespecs.**StringTableSpec**

Bases: object

**__init__**(*byte_width=8*)

**append_args**(*args*, *add_null=True*)

**append_env**(*env*, *add_null=True*)

**add_string**(*string*)

**add_pointer**(*pointer*)

**add_null**()

**dump**(*state*, *end_addr*, *align=16*)

## 10.46 angr.utils

angr.utils.**is_pyinstaller**()

Detect if we are currently running as a PyInstaller-packaged program.

**Return type**
bool

**Returns**
True if we are running as a PyInstaller-packaged program. False if we are running in Python directly (e.g., development mode).

angr.utils.**looks_like_sql**(*s*)

Determine if string *s* looks like an SQL query.

**Parameters**
**s** (str) – The string to detect.

**Return type**
bool

**Returns**
True if the string looks like an SQL, False otherwise.

angr.utils.**timethis**(*func*)

### Submodules

*ail*

continues on next page

Table 115 – continued from previous page

| | |
|---|---|
| *ail_serialization* | Typed protobuf pack/parse helpers for AIL-typed containers. |
| *algo* | |
| *balancer* | |
| *bits* | |
| *constants* | |
| *cowdict* | |
| *cpp* | |
| *doms* | |
| *dynamic_dictlist* | |
| *endness* | |
| *enums_conv* | |
| *env* | |
| *formatting* | |
| *funcid* | |
| *go_runtime* | Identification of Go runtime functions that never return. |
| *graph* | |
| *hashing* | |
| *ins_addr_list* | |
| *json_utils* | JSON encode/decode wrappers that use msgspec when available (CPython only), falling back to the standard library json module otherwise. |
| *lazy_import* | |
| *library* | |
| *loader* | |
| *mp* | |
| *orderedset* | |
| *smart_cache* | |
| *ssa* | |
| *strings* | |
| *tagged_interval_map* | |
| *timing* | |
| *types* | |
| *vex* | |

### 10.46.1 angr.utils.ail

angr.utils.ail.**is_phi_assignment**(*stmt*)

**Return type**
bool

**Parameters**
**stmt** (Statement)

angr.utils.ail.**is_scalar_int_convert**(*expr*)

True for a plain integer width conversion: not lane-wise (vector_count is None) and neither side is floating point. Only these converts admit the usual integer rewrites (folding cascaded conversions, masking, etc.).

**Return type**
bool

**Parameters**
**expr** (Convert)

class angr.utils.ail.**HasExprWalker**

Bases: *AILBlockViewer*

Test if any expressions in exprs_to_check is used in another AIL expression.

**__init__**(*exprs_to_check*)

**Parameters**
**exprs_to_check** (*set[Expression]*)

**Return type**
None

angr.utils.ail.**is_head_controlled_loop_block**(*block*)

Determine if the block is a "head-controlled loop." A head-controlled loop (for the lack of a better term) is a single-block that contains a conditional jump towards the beginning of the block. This conditional jump controls whether the loop body (the remaining statements after the conditional jump) will be executed or not. It is usually the result of lifting rep stosX instructions on x86 and amd64.

A head-controlled loop block looks like the following (lifted from rep stosq qword ptr [rdi], rax):

## Block 4036df 00 | 0x4036df | LABEL_4036df: 01 | 0x4036df | vvar_27{reg 72} = @64b [] 02 | 0x4036df | vvar_28{reg 24} = @64b [] 03 | 0x4036df | t1 = rcx<8> 04 | 0x4036df | t4 = (t1 == 0x0<64>) 05 | 0x4036df | if (t4) { Goto 0x4036e2<64> } else { Goto 0x4036df<64> } 06 | 0x4036df | t5 = (t1 - 0x1<64>) 07 | 0x4036df | rcx<8> = t5 08 | 0x4036df | t7 = d<8> 09 | 0x4036df | t6 = (t7 << 0x3<8>) 10 | 0x4036df | t2 = rax<8> 11 | 0x4036df | t3 = rdi<8> 12 | 0x4036df | STORE(addr=t3, data=t2, size=8, endness=Iend_LE, guard=None) 13 | 0x4036df | t8 = (t3 + t6) 14 | 0x4036df | rdi<8> = t8

Where statement 5 is the conditional jump that controls the execution of the remaining statements of this block.

**Parameters**
**block** (Block) – An AIL block.

**Return type**
bool

**Returns**
True if the block represents a head-controlled loop block, False otherwise.

angr.utils.ail.**extract_partial_expr**(*base_expr*, *off*, *size*, *ail_manager*, *byte_width=8*)

**Return type**
*Expression*

**Parameters**

- **base_expr** (Expression)
- **off** (*int*)
- **size** (*int*)
- **byte_width** (*int*)

angr.utils.ail.**is_expr_used_as_reg_base_value**(*stmt*, *expr*, *srda*)

Determine if the expression *expr* is used as the base value of an assignment of a full register in *stmt*.

This method returns True if the following conditions hold: - The statement is an assignment to a reg vvar A; - The src of the assignment statement is a bitwise-or expression; - One of the operands of the src expr is the high bits of *expr*; - *expr* is a phi var that relies on reg vvar A.

**Return type**
bool

**Parameters**

- **stmt** (Statement)
- **expr** (Expression)
- **srda** (SRDAModel)

### 10.46.2 angr.utils.ail_serialization

Typed protobuf pack/parse helpers for AIL-typed containers.

AIL leaves (Block / Statement / Expression) serialize through their native `to_bytes()` methods (postcard, implemented in Rust); the helpers here only encode the Python container structure around them using the typed messages in *angr.protos.ail_types_pb2*. There is no generic fallback: any value shape not covered by the schema raises `TypeError`.

angr.utils.ail_serialization.**simvar_to_bytes_polymorphic**(*v*)

Polymorphic SimVariable encoding: `b"<ClassName>\0<proto bytes>"`.

**Return type**
bytes

**Parameters**
**v** (SimVariable)

angr.utils.ail_serialization.**simvar_from_bytes_polymorphic**(*b*)

**Return type**
*SimVariable*

**Parameters**
**b** (bytes)

**class** angr.utils.ail_serialization.**BlockPool**

Bases: object

A shared pool of `Block.to_bytes()` payloads for deduplicating byte-identical blocks across graphs.

**__init__**()

**Return type**
None

**payloads:** **list**[*bytes*]

**add**(*block*)

**Return type**
int

angr.utils.ail_serialization.**pack_graph**(*graph*, *pool=None*)

Encode a DiGraph of ailment Blocks. Node identity is preserved through per-graph block indices. When `pool` is given, block payloads are deduplicated into it and the message stores pool refs instead of inline payloads.

**Return type**
*AilGraph*

**Parameters**

- **graph** (*DiGraph*)
- **pool** (BlockPool | *None*)

angr.utils.ail_serialization.**parse_graph**(*msg*, *pool_payloads=None*)

**Return type**
DiGraph

**Parameters**
**msg** (AilGraph)

angr.utils.ail_serialization.**pack_arg_vvars**(*arg_vvars*)

**Return type**
*ArgVVars*

**Parameters**
**arg_vvars** (*dict[int, tuple[Any, Any]]*)

angr.utils.ail_serialization.**parse_arg_vvars**(*msg*)

**Return type**
dict[int, tuple[Any, Any]]

**Parameters**
**msg** (ArgVVars)

angr.utils.ail_serialization.**pack_ite_exprs**(*ite_exprs*)

**Return type**
*IteExprs*

**Parameters**
**ite_exprs** (*set[tuple[int, Any]]*)

angr.utils.ail_serialization.**parse_ite_exprs**(*msg*)

**Return type**
set[tuple[int, Any]]

**Parameters**
**msg** (IteExprs)

angr.utils.ail_serialization.**pack_static_vvars**(*static_vvars*)

**Return type**
*StaticVVars*

**Parameters**
**static_vvars** (*dict[int, Any]*)

angr.utils.ail_serialization.**parse_static_vvars**(*msg*)

**Return type**
dict[int, Any]

**Parameters**
**msg** (StaticVVars)

angr.utils.ail_serialization.**pack_static_buffers**(*static_buffers*)

**Return type**
*StaticBuffers*

**Parameters**
**static_buffers** (*dict[str, Any]*)

angr.utils.ail_serialization.**parse_static_buffers**(*msg*)

**Return type**
dict[str, Any]

**Parameters**
**msg** (StaticBuffers)

### 10.46.3 angr.utils.algo

angr.utils.algo.**binary_insert**(*lst*, *elem*, *key*, *lo=0*, *hi=None*)

Insert an element into a sorted list, and keep the list sorted.

The major difference from bisect.bisect_left is that this function supports a key method, so user doesn't have to create the key array for each insertion.

**Parameters**

- **lst** (list | deque) – The list. Must be pre-ordered.
- **element** – An element to insert into the list.
- **key** (Callable) – A method to get the key for each element in the list.
- **lo** (int) – Lower bound of the search.
- **hi** (int | None) – Upper bound of the search.
- **elem** (*Any*)

**Return type**
None

**Returns**
None

### 10.46.4 angr.utils.balancer

**exception** angr.utils.balancer.**BalancerError**

Bases: *AngrError*

Base class for balancer errors.

**exception** angr.utils.balancer.**BalancerUnsatError**

Bases: *BalancerError*

Exception raised when the balancer determines the constraints are unsatisfiable.

**class** angr.utils.balancer.**Balancer**

Bases: object

The Balancer is an equation redistributor. The idea is to take an AST and rebalance it to, for example, isolate unknown terms on one side of an inequality.

**__init__**(*c*)

**Parameters**
**c** (*Bool*)

**property compat_ret:** tuple[bool, list[tuple[BV, BV]]]

**property replacements:** list[tuple[BV, BV]]

```
comparison_info = {'SGE': (False, True, False), 'SGT': (False, False, False), 'SLE':
(True, True, False), 'SLT': (True, False, False), 'UGE': (False, True, True), 'UGT':
(False, False, True), 'ULE': (True, True, True), 'ULT': (True, False, True)}
```

angr.utils.balancer.**constraint_to_si**(*expr*)

Convert a constraint to SI if possible.

**Parameters**
**expr** (Bool)

**Return type**
tuple[bool, list[tuple[BV, BV]]]

**Returns**

### 10.46.5 angr.utils.bits

angr.utils.bits.**truncate_bits**(*value*, *nbits*)

Truncate *value* to *nbits*.

For example: truncate_bits(0x1234, 8) -> 0x34

**Return type**
int

**Parameters**

- **value** (*int*)
- **nbits** (*int*)

angr.utils.bits.**ffs**(*x*)

**Return type**
int

**Parameters**
**x** (*int*)

angr.utils.bits.**sign_extend**(*value*, *bits*)

**Return type**
int

**Parameters**

- **value** (*int*)
- **bits** (*int*)

angr.utils.bits.**zeroextend_on_demand**(*op0*, *op1*)

ZeroExtend op1 if the size of op1 is smaller than the size of op0. Otherwise, return op1.

**Return type**
BV

**Parameters**

- **op0** (*BV*)
- **op1** (*BV*)

angr.utils.bits.**s2u**(*s*, *bits*)

angr.utils.bits.**u2s**(*u*, *bits*)

### 10.46.6 angr.utils.constants

angr.utils.constants.**MAGIC_CONSTANTS = frozenset({195948557, 2343432205, 2880289470, 3131961357, 3135097598, 3405691582, 3405697037, 3735927486, 3735928559, 3735929054, 3735936685, 4207869677, 4276215469, 4277009102})**

Well-known "magic" constants that are universally recognized in hexadecimal.

angr.utils.constants.**is_alignment_mask**(*n*)

angr.utils.constants.**should_use_hex**(*value*, *bits=None*)

Heuristically decide whether an integer constant reads better in hexadecimal than in decimal in decompiled output.

The rules are checked in priority order:

Hexadecimal (returns `True`):

0. Small negative values (`-255 <= value < 0`) are always shown in decimal.
1. The value is a well-known "magic" constant (e.g. `0xdeadbeef`); see *MAGIC_CONSTANTS*.
2. The value is a known alignment mask; see *is_alignment_mask()*.
3. The binary representation contains a run of **>= 8** consecutive 1 bits – typical of sub-word bitmasks (e.g. `0xff`, `0xfff`).
4. The binary representation is a short bit pattern (period 2, 3, 4, or 8) repeated to fill a byte-aligned width – typical of mask/flag constants (e.g. `0x55`/`0xaaaa` for `0101...`, `0x1111`, `0xabab`).
5. The value is a single set bit (power of two) that is **>= 256** – a bit flag (e.g. `0x100`, `0x4000`).

Decimal (returns `False`):

6. The decimal representation of `abs(value)` contains a run of **> 3** (i.e. >= 4) identical digits (e.g. `10000`, `11111`, `1000000`).
7. The value is a nonzero multiple of 1000 – a "round" human-authored decimal number (e.g. `5000`, `20000`).
8. The magnitude is small (`abs(value) <= 9`) – e.g. small loop counters and offsets.

Hexadecimal (weak signal, returns `True`):

9. The low byte is zero and the value is > `0xff` – a "round" hex number (e.g. `0x1200`).
10. The value is a common error code (e.g. `0xc0000005`) in a 32- or 64-bit context.

Otherwise the value is shown in decimal (returns `False`).

**Parameters**

- **value** (int) – The integer value to format.
- **bits** (int | None) – The bit-width of the value's type. Used to normalize negative values to their unsigned form. If `None`, the value's own bit length is used.

**Return type**
bool

**Returns**
`True` if the value should be displayed in hexadecimal, `False` for decimal.

### 10.46.7 angr.utils.cowdict

angr.utils.cowdict.**merge_candidate_keys**(*a*, *b*)

Return the keys that may differ between two COW chain maps which (typically) descend from a common ancestor and therefore share a common suffix of layer objects.

A key whose binding lives entirely in the shared suffix – an identical `maps` entry (by identity) that is not shadowed by a deletion in either map – resolves to the same value in both maps, so unioning them is always a no-op and the key is omitted. The result is the union of the keys held in each map's non-shared "head" layers, plus the keys either map has logically deleted: a deleted key may still be bound in the other map's shared suffix and therefore needs to be resurrected by a merge.

**Return type**
set

**Parameters**

- **a** (ChainMapCOW)
- **b** (ChainMapCOW)

**class** angr.utils.cowdict.**ChainMapCOW**

Bases: ChainMap, Generic

Implements a copy-on-write version of ChainMap that supports auto-collapsing.

Tracks logically deleted keys via a _deleted set so that pop() and del work correctly even when keys live in parent maps.

Performance note: We deliberately re-implement the ChainMap walk for better performance. `ChainMap.__getitem__` raises a `KeyError` for every layer that misses, and `ChainMap.get` walks the whole chain twice (`__contains__` and `__getitem__`). We avoid both by inlining `if key in mapping`.

We may revisit the implementation in the future when the performance of ChainMap improves.

**maps**

**dirty**

**collapse_threshold**

**clean**()

**Return type**
*ChainMapCOW*[TypeVar(K), TypeVar(V)]

**class** angr.utils.cowdict.**DefaultChainMapCOW**

Bases: *ChainMapCOW*, Generic

Implements a copy-on-write version of ChainMap with default values that supports auto-collapsing.

**default_factory**

**clean**()

**Return type**
*DefaultChainMapCOW*[TypeVar(K), TypeVar(V)]

### 10.46.8 angr.utils.cpp

angr.utils.cpp.**is_cpp_funcname_ctor**(*name*)

Check if a demangled C++ function name is a constructor.

**Parameters**
**name** (str) – The demangled C++ function name.

**Return type**
bool

**Returns**
True if the function name is a constructor, False otherwise.

### 10.46.9 angr.utils.doms

**class** angr.utils.doms.**IncrementalDominators**

Bases: object

This class allows for incrementally updating dominators and post-dominators for graphs. The graph must only be modified by replacing nodes, not adding nodes or edges.

**__init__**(*graph*, *start*, *post=False*)

**Parameters**
- **graph** (*DiGraph*)
- **post** (*bool*)

**init_doms**()

**Return type**
dict[Any, Any]

**init_dfs**()

**Return type**
dict[Any, set[Any]]

**graph_updated**(*new_node*, *replaced_nodes*, *replaced_head*)

**Parameters**
- **new_node** (*Any*)
- **replaced_nodes** (*set[Any]*)
- **replaced_head** (*Any*)

**idom**(*node*)

Get the immediate dominator of a given node.

**Return type**
Any | None

**Parameters**
**node** (*Any*)

**df**(*node*)

Generate the dominance frontier of a node.

**Return type**
set[Any]

**Parameters**
**node** (*Any*)

**dominates**(*dominator_node*, *node*)

Tests if dominator_node dominates (or post-dominates) node.

**Return type**
bool

**Parameters**

- **dominator_node** (*Any*)
- **node** (*Any*)

### 10.46.10 angr.utils.dynamic_dictlist

**class** angr.utils.dynamic_dictlist.**DynamicDictList**

Bases: Generic

A list-like container class that internally uses dicts to store values when the number of values is less than the threshold *LIST2DICT_THRESHOLD*. Keys must be ints.

The default thresholds are determined according to experiments described at https://github.com/angr/angr/pull/3471#issuecomment-1236515950.

**__init__**(*max_size=None*, *content=None*)

**Parameters**

- **max_size** (*int | None*)
- **content** (DynamicDictList | *dict[int, VT]* | *list[VT]* | *None*)

**list_content:** list[VT] | None

**max_size**

**dict_content:** dict[int, VT] | None

**real_length**()

**Return type**
int

### 10.46.11 angr.utils.endness

angr.utils.endness.**ail_const_to_be**(*expr*, *endness*)

**Return type**
*Const*

**Parameters**

- **expr** (Const)
- **endness** (*str*)

### 10.46.12 angr.utils.enums_conv

angr.utils.enums_conv.**cfg_jumpkind_to_pb**(*jk*)

angr.utils.enums_conv.**func_edge_type_to_pb**(*jk*)

angr.utils.enums_conv.**cfg_jumpkind_from_pb**(*pb*)

angr.utils.enums_conv.**func_edge_type_from_pb**(*pb*)

### 10.46.13 angr.utils.env

angr.utils.env.**is_pyinstaller**()

Detect if we are currently running as a PyInstaller-packaged program.

**Return type**
bool

**Returns**
True if we are running as a PyInstaller-packaged program. False if we are running in Python directly (e.g., development mode).

### 10.46.14 angr.utils.formatting

angr.utils.formatting.**setup_terminal**()

Check if we are running in a TTY. If so, make sure the terminal supports ANSI escape sequences. If not, disable colorized output. Sets global *ansi_color_enabled* to True if colorized output should be enabled by default.

angr.utils.formatting.**ansi_color**(*s*, *color*)

Colorize string *s* by wrapping in ANSI escape sequence for given *color*.

This function does not consider whether escape sequences are functional or not; it is up to the caller to determine if its appropriate. Check global *ansi_color_enabled* value in this module.

**Return type**
str

**Parameters**

- **s** (*str*)
- **color** (*str | None*)

angr.utils.formatting.**add_edge_to_buffer**(*buf*, *ref*, *start*, *end*, *formatter=None*, *dashed=False*, *ascii_only=None*)

Draw an edge by adding Unicode box and arrow glyphs to beginning of each line in a list of lines.

**Parameters**

- **buf** (Sequence[str]) – Output buffer, used to render formatted edges.
- **ref** (Sequence[str]) – Reference buffer, used to calculate edge depth.
- **start** (int) – Start line.
- **end** (int) – End line, where arrow points.
- **formatter** (Callable[[str], str] | None) – Optional callback function used to format the edge before writing it to output buffer.
- **dashed** (bool) – Render edge line dashed instead of solid.

- **ascii_only** (bool | None) – Render edge using ASCII characters only. If unspecified, guess by stdout encoding.

**Returns**

### 10.46.15 angr.utils.funcid

angr.utils.funcid.**is_function_security_check_cookie**(*func*, *project*, *security_cookie_addr*)

**Return type**
bool

**Parameters**
**security_cookie_addr** (*int*)

angr.utils.funcid.**is_function_security_check_cookie_strict**(*func*, *project*)

**Return type**
tuple[bool, int | None]

**Parameters**
**func** (Function)

angr.utils.funcid.**is_function_security_init_cookie**(*func*, *project*, *security_cookie_addr*)

**Return type**
bool

**Parameters**

- **func** (Function)
- **security_cookie_addr** (*int | None*)

angr.utils.funcid.**is_function_security_init_cookie_win8**(*func*, *project*, *security_cookie_addr*)

**Return type**
bool

**Parameters**

- **func** (Function)
- **security_cookie_addr** (*int*)

angr.utils.funcid.**is_function_likely_security_init_cookie**(*func*)

Conducts a fuzzy match for security_init_cookie function.

**Return type**
bool

**Parameters**
**func** (Function)

### 10.46.16 angr.utils.go_runtime

Identification of Go runtime functions that never return.

angr has no built-in knowledge of the Go runtime, so the two families of stubs that the Go compiler emits at essentially every call site – the goroutine stack-growth stub (`runtime.morestack`) and the bounds-check panic stubs – are recovered as returning functions. That attaches a large amount of unreachable code to every Go function: a fake self-recursive tail after each `morestack` call site, and one dead panic branch per bounds check.

Two identification strategies are provided, because the interesting targets are usually stripped:

- by name, when symbols (or a `.gopclntab`-derived symbol table) are available;
- by shape, which is what actually carries stripped binaries.

A false positive here silently deletes real code, so every shape-based rule below requires either overwhelming corroboration across the binary or an assembly signature that no compiler emits for ordinary code.

angr.utils.go_runtime.**GO_NORETURN_NAMES**: frozenset[str] = frozenset({'os.Exit', 'runtime.Goexit', 'runtime.abort', 'runtime.badctxt', 'runtime.badmcall', 'runtime.badmcall2', 'runtime.badreflectcall', 'runtime.exit', 'runtime.exitThread', 'runtime.exitsyscall0', 'runtime.fatal', 'runtime.fatalpanic', 'runtime.fatalthrow', 'runtime.goPanicExtendIndex', 'runtime.goPanicExtendIndexU', 'runtime.goPanicExtendSlice3Acap', 'runtime.goPanicExtendSlice3AcapU', 'runtime.goPanicExtendSlice3Alen', 'runtime.goPanicExtendSlice3AlenU', 'runtime.goPanicExtendSlice3B', 'runtime.goPanicExtendSlice3BU', 'runtime.goPanicExtendSlice3C', 'runtime.goPanicExtendSlice3CU', 'runtime.goPanicExtendSliceAcap', 'runtime.goPanicExtendSliceAcapU', 'runtime.goPanicExtendSliceAlen', 'runtime.goPanicExtendSliceAlenU', 'runtime.goPanicExtendSliceB', 'runtime.goPanicExtendSliceBU', 'runtime.goPanicExtendSliceConvert', 'runtime.goPanicIndex', 'runtime.goPanicIndexU', 'runtime.goPanicSlice3Acap', 'runtime.goPanicSlice3AcapU', 'runtime.goPanicSlice3Alen', 'runtime.goPanicSlice3AlenU', 'runtime.goPanicSlice3B', 'runtime.goPanicSlice3BU', 'runtime.goPanicSlice3C', 'runtime.goPanicSlice3CU', 'runtime.goPanicSliceAcap', 'runtime.goPanicSliceAcapU', 'runtime.goPanicSliceAlen', 'runtime.goPanicSliceAlenU', 'runtime.goPanicSliceB', 'runtime.goPanicSliceBU', 'runtime.goPanicSliceConvert', 'runtime.goexit', 'runtime.goexit0', 'runtime.goexit1', 'runtime.gogo', 'runtime.gopanic', 'runtime.goschedImpl', 'runtime.main', 'runtime.mcall', 'runtime.morestack', 'runtime.morestack_noctxt', 'runtime.morestackc', 'runtime.panicBounds', 'runtime.panicBounds32', 'runtime.panicBounds64', 'runtime.panicExtendIndex', 'runtime.panicExtendIndexU', 'runtime.panicExtendSlice3Acap', 'runtime.panicExtendSlice3AcapU', 'runtime.panicExtendSlice3Alen', 'runtime.panicExtendSlice3AlenU', 'runtime.panicExtendSlice3B', 'runtime.panicExtendSlice3BU', 'runtime.panicExtendSlice3C', 'runtime.panicExtendSlice3CU', 'runtime.panicExtendSliceAcap', 'runtime.panicExtendSliceAcapU', 'runtime.panicExtendSliceAlen', 'runtime.panicExtendSliceAlenU', 'runtime.panicExtendSliceB', 'runtime.panicExtendSliceBU', 'runtime.panicExtendSliceConvert', 'runtime.panicIndex', 'runtime.panicIndexU', 'runtime.panicSlice3Acap', 'runtime.panicSlice3AcapU', 'runtime.panicSlice3Alen', 'runtime.panicSlice3AlenU', 'runtime.panicSlice3B', 'runtime.panicSlice3BU', 'runtime.panicSlice3C', 'runtime.panicSlice3CU', 'runtime.panicSliceAcap', 'runtime.panicSliceAcapU', 'runtime.panicSliceAlen', 'runtime.panicSliceAlenU', 'runtime.panicSliceB', 'runtime.panicSliceBU', 'runtime.panicSliceConvert', 'runtime.panicdivide', 'runtime.panicdottypeE', 'runtime.panicdottypeI', 'runtime.panicfloat', 'runtime.panicmakeslicecap', 'runtime.panicmakeslicelen', 'runtime.panicmem', 'runtime.panicmemAddr', 'runtime.panicnildottype', 'runtime.panicoverflow', 'runtime.panicshift', 'runtime.panicunsafeslicelen', 'runtime.panicunsafeslicelen1', 'runtime.panicunsafeslicenilptr', 'runtime.panicunsafeslicenilptr1', 'runtime.panicunsafestringlen', 'runtime.panicunsafestringlen1', 'runtime.panicunsafestringnilptr', 'runtime.panicunsafestringnilptr1', 'runtime.panicwrap', 'runtime.park_m', 'runtime.schedule', 'runtime.sigpanic', 'runtime.sigpanic0', 'runtime.throw'})

Go runtime (and a few closely related standard library) functions that never transfer control back to the instruction following their call site. `runtime.morestack` and friends do resume the caller, but at its entry point rather than at the return address, so they do not "return" in the sense CFG recovery cares about.

Deliberately excluded because they can fall through to their caller despite the suggestive names: `runtime.mexit` (returns when running on an OS-provided stack, and with it `runtime.mstart0` and `runtime.mstart`), `runtime.badsystemstack`, `runtime.badmorestackg0`, `runtime.badmorestackgsignal` (all only print), `runtime.systemstack`, `runtime.panicCheck1`, `runtime.panicCheck2`, `runtime.startpanic_m`, `runtime.dopanic_m`.

angr.utils.go_runtime.**has_go_hint**(*project*)

Cheap test for "this might be a Go binary", to keep the (much more expensive) LanguageDetector off the vast majority of binaries.

**Return type**
bool

**Parameters**
**project** (Project)

angr.utils.go_runtime.**normalize_go_func_name**(*name*)

Strip the ABI wrapper suffix the Go linker appends to duplicated symbols.

**Return type**
str

**Parameters**
**name** (*str*)

angr.utils.go_runtime.**is_go_noreturn_name**(*name*)

**Return type**
bool

**Parameters**
**name** (*str*)

angr.utils.go_runtime.**find_go_noreturn_functions**(*project*, *kb=None*, *use_names=True*)

Identify Go runtime functions in `project` that never return.

The caller is responsible for having established that this is a Go binary.

**Parameters**

- **use_names** (bool) – Consult symbol names. Set to False to exercise the shape-based path that stripped binaries depend on.
- **project** (Project)

**Return type**
dict[int, str]

**Returns**
A mapping from function address to a short description of the evidence.

### 10.46.17 angr.utils.graph

angr.utils.graph.**shallow_reverse**(*g*)

Make a shallow copy of a directional graph and reverse the edges. This is a workaround to solve the issue that one cannot easily make a shallow reversed copy of a graph in NetworkX 2, since networkx.reverse(copy=False) now returns a GraphView, and GraphViews are always read-only.

**Parameters**
**g** (*networkx.DiGraph*) – The graph to reverse.

**Returns**
A new networkx.DiGraph that has all nodes and all edges of the original graph, with edges reversed.

angr.utils.graph.**inverted_idoms**(*graph*, *end_node=None*)

Invert the given graph and generate the immediate dominator tree on the inverted graph. This is useful for computing post-dominators.

**Parameters**
**graph** – The graph to invert and generate immediate dominator tree for.

**Returns**
A tuple of the inverted graph and the immediate dominator tree.

angr.utils.graph.**to_acyclic_graph**(*graph*, *node_order=None*, *loop_heads=None*)

Convert a given DiGraph into an acyclic graph.

**Parameters**

- **graph** – The graph to convert.
- **ordered_nodes** – A list of nodes sorted in a topological order.
- **loop_heads** – A list of known loop head nodes.

**Returns**
The converted acyclic graph.

angr.utils.graph.**dfs_back_edges**(*graph*, *start_node*, ***, *visit_all_nodes=False*, *visited=None*)

Perform an iterative DFS traversal of the graph, returning back edges.

**Parameters**

- **graph** – The graph to traverse.
- **start_node** – The node where to start the traversal.

**Returns**
An iterator of 'backward' edges.

angr.utils.graph.**subgraph_between_nodes**(*graph*, *source*, *frontier*, *include_frontier=False*)

For a directed graph, return a subgraph that includes all nodes going from a source node to a target node.

**Parameters**

- **graph** (*networkx.DiGraph*) – The directed graph.
- **source** – The source node.
- **frontier** (*list*) – A collection of target nodes.
- **include_frontier** (*bool*) – Should nodes in frontier be included in the subgraph.

**Returns**
A subgraph.

**Return type**
networkx.DiGraph

angr.utils.graph.**dominates**(*idom*, *dominator_node*, *node*)

**Return type**
bool

**Parameters**

- **idom** (*dict[T, T]*)
- **dominator_node** (*T*)
- **node** (*T*)

angr.utils.graph.**compute_dominance_intervals**(*idom*, *head*)

Compute entry/exit (Euler tour) intervals over the dominator tree described by idom, so that dominance queries answer in O(1) via dominates_by_intervals() instead of dominates()'s O(tree depth) chain walk. Only nodes present in idom (i.e., reachable from the head) get an interval.

**Return type**
dict[TypeVar(T), tuple[int, int]]

**Parameters**

- **idom** (*dict[T, T]*)
- **head** (*T*)

angr.utils.graph.**dominates_by_intervals**(*intervals*, *dominator_node*, *node*)

O(1) equivalent of dominates() over intervals from compute_dominance_intervals(). Node objects must be truthy (all AIL and structurer nodes are), which is the only case where dominates() behaves differently.

**Return type**
bool

**Parameters**

- **intervals** (*dict[T, tuple[int, int]]*)
- **dominator_node** (*T*)
- **node** (*T*)

angr.utils.graph.**compute_dominance_frontier**(*graph*, *domtree*)

Compute a dominance frontier based on the given post-dominator tree.

This implementation is based on figure 2 of paper An Efficient Method of Computing Static Single Assignment Form by Ron Cytron, etc.

**Parameters**

- **graph** – The graph where we want to compute the dominance frontier.
- **domtree** – The dominator tree

**Returns**
A dict of dominance frontier

**class** angr.utils.graph.**TemporaryNode**

Bases: object

A temporary node.

Used as the start node and end node in post-dominator tree generation. Also used in some test cases.

**__init__**(*label*)

**class** angr.utils.graph.**ContainerNode**

Bases: Generic

A container node.

Only used in dominator tree generation. We did this so we can set the index property without modifying the original object.

__init__(*obj*)

**Parameters**
**obj** (*T*)

**index:** int | None

**property obj:** T

class angr.utils.graph.**Dominators**

Bases: Generic

Describes dominators in a graph.

__init__(*graph*, *entry_node*, *successors_func=None*, *reverse=False*)

**dom:** networkx.DiGraph[T]

class angr.utils.graph.**PostDominators**

Bases: *Dominators*, Generic

Describe post-dominators in a graph.

__init__(*graph*, *entry_node*, *successors_func=None*)

**property post_dom:** networkx.DiGraph[T]

class angr.utils.graph.**SCCPlaceholder**

Bases: object

Describes a placeholder for strongly-connected-components in a graph.

__init__(*scc_id*, *addr*)

**scc_id**

**addr**

class angr.utils.graph.**GraphUtils**

Bases: object

A helper class with some static methods and algorithms implemented, that in fact, might take more than just normal CFGs.

**static find_merge_points**(*function_addr*, *function_endpoints*, *graph*)

Given a local transition graph of a function, find all merge points inside, and then perform a quasi-topological sort of those merge points.

A merge point might be one of the following cases: - two or more paths come together, and ends at the same address. - end of the current function

**Parameters**

- **function_addr** (*int*) – Address of the function.
- **function_endpoints** (*list*) – Endpoints of the function. They typically come from Function.endpoints.
- **graph** (*networkx.DiGraph*) – A local transition graph of a function. Normally it comes from Function.graph.

**Returns**
A list of ordered addresses of merge points.

**Return type**
list

**static find_widening_points**(*function_addr*, *function_endpoints*, *graph*)

Given a local transition graph of a function, find all widening points inside.

Correctly choosing widening points is very important in order to not lose too much information during static analysis. We mainly consider merge points that has at least one loop back edges coming in as widening points.

**Parameters**

- **function_addr** (*int*) – Address of the function.
- **function_endpoints** (*list*) – Endpoints of the function, typically coming from Function.endpoints.
- **graph** (*networkx.DiGraph*) – A local transition graph of a function, normally Function.graph.

**Returns**
A list of addresses of widening points.

**Return type**
list

**static dfs_postorder_nodes_deterministic**(*graph*, *source*)

**Parameters**

- **graph** (*networkx.DiGraph[T]*)
- **source** (*T*)

**Return type**
Iterator[T]

**static dfs_postorder_nodes_deterministic_multi**(*graph*, *sources*)

Deterministic post-order DFS covering multiple source nodes with a single shared `visited` set. Equivalent to `dfs_postorder_nodes_deterministic` over a synthetic super-source connected to every node in `sources` (sources are traversed in `_sort_node` order), but without mutating the graph – so it works directly on a zero-copy overlay view. Use it for multi-entry acyclic graphs.

**Parameters**

- **graph** (*networkx.DiGraph[T]*)
- **sources** (*Iterable[T]*)

**Return type**
Iterator[T]

**static reverse_post_order_sort_nodes**(*graph*, *nodes=None*)

Sort a given set of nodes in reverse post ordering.

**Parameters**

- **graph** (*networkx.DiGraph*) – A local transition graph of a function.
- **nodes** (*iterable*) – A collection of nodes to sort.

**Returns**
A list of sorted nodes.

**Return type**
list

**static sort_node**(*node*)

A sorter to make a deterministic order of nodes. The key is a total order: nodes sharing an address (duplicated blocks) are separated by their idx instead of falling back on insertion order.

**Return type**
tuple[int, int, int, str]

**static sort_edge**(*edge*)

A sorter to make a deterministic order of edges.

**static quasi_topological_sort_nodes**(*graph*, *nodes=None*, *loop_heads=None*, *panic_mode_threshold=3000*)

Sort a given set of nodes from a graph based on the following rules:

# - if A -> B and not B -> A, then we have A < B # - if A -> B and B -> A, then the ordering is undefined

Following the above rules gives us a quasi-topological sorting of nodes in the graph. It also works for cyclic graphs.

**Parameters**

- **graph** (DiGraph) – A local transition graph of the function.
- **nodes** (list | None) – A list of nodes to sort. None if you want to sort all nodes inside the graph.
- **loop_heads** (list | None) – A list of nodes that should be treated loop heads.
- **panic_mode_threshold** (int) – Threshold of nodes in an SCC to begin aggressively removing edges.

**Return type**
list

**Returns**
A list of ordered nodes.

**static loop_nesting_forest**(*graph*, *start_node*)

Generates the loop-nesting forest for the provided directional graph. This is *not* the algorithm proposed by Ramalingam.

**Parameters**

- **graph** (DiGraph) – the graph to generate the loop-nesting forest for.
- **start_node** – the node to start traversing the graph from.

**Return type**
OrderedDict[Any, DiGraph]

**Returns**
An ordered dict of loop heads to their corresponding loop nodes.

**class** angr.utils.graph.**LinkedListNode**

Bases: Generic

A node in a linked list.

__init__(*v*)

**Parameters**
**v** (*T*)

**v: T**

**next: *LinkedListNode* | None**

**prev: *LinkedListNode* | None**

class angr.utils.graph.**DirectedGraphHelper**

Bases: Generic

A helper class for accelerating some graph algorithms on directed graphs.

_node_order keeps a dictionary of nodes and their order in a quasi-topological sort of the region full graph (graph_with_successors). _generate_node_order() initializes this dictionary. we then update this dictionary when new nodes are created.

We do not populate _node_order when working on acyclic graphs because it's not used for acyclic graphs.

__init__(*graph*, *cyclic_graph*, *head*)

**reset**()

**dfs_postorder_nodes_deterministic**(*source*)

Post-order DFS traversal of the directed graph using cache. Avoids acyclic-graph conversion or full-graph traversal when cache is available.

**Return type**
Iterator[TypeVar(T)]

**Parameters**
**source** (*T*)

**replace_node**(*old_node*, *new_node*)

Update cache when a node is replaced in the graph.

**Parameters**

- **old_node** (*T*)
- **new_node** (*T*)

**replace_nodes**(*old_node_0*, *old_node_1*, *new_node*)

Update cache when multiple nodes are replaced by a single node in the graph.

**Return type**
None

**Parameters**

- **old_node_0** (*T*)
- **old_node_1** (*T*)
- **new_node** (*T*)

**remove_node**(*node*)

Update cache when a node is removed from the graph.

**Parameters**
**node** (*T*)

**add_node_successor**(*node*, *successor*)

**Return type**
None

**Parameters**

- **node** (*T*)
- **successor** (*T*)

**get_node_order**()

**Return type**
dict[TypeVar(T), int]

**to_acyclic_by_order**(*graph*)

**Parameters**
**graph** (*RegionOverlayGraph[T]*)

**Return type**
networkx.DiGraph[T]

**sort_nodes_by_order**(*nodes*)

**Return type**
list[TypeVar(T)]

**Parameters**
**nodes** (*list[T]*)

**loop_heads**()

**Return type**
set[TypeVar(T)]

### 10.46.18 angr.utils.hashing

angr.utils.hashing.**stable_hash**(*t*)

**Return type**
int

**Parameters**
**t** (*tuple*)

### 10.46.19 angr.utils.ins_addr_list

**class** angr.utils.ins_addr_list.**InsAddrList**

Bases: object

A memory-efficient replacement for list[int] that stores instruction addresses as a base address plus a bytestring of instruction sizes (one byte per instruction).

Address reconstruction: addr[i] = base_addr + sum(ins_sizes[0:i])

For compatibility, this class exposes a list-like interface (iteration, indexing, length, membership testing, concatenation, etc.).

**__init__**(*base_addr=0*, *ins_sizes=b''*)

**Parameters**

- **base_addr** (*int*)
- **ins_sizes** (*bytes | bytearray | list[int]*)

**classmethod from_addr_list**(*addrs*)

Construct an InsAddrList from a list/sequence of absolute instruction addresses.

**Return type**
*InsAddrList*

**property base_addr: int**

**property ins_sizes: bytes**

**extend**(*other*)

**Return type**
None

**Parameters**
**other** (InsAddrList)

**index**(*value*, *start=0*, *stop=None*)

**count**(*value*)

**Return type**
int

**copy**()

**Return type**
*InsAddrList*

### 10.46.20 angr.utils.json_utils

JSON encode/decode wrappers that use msgspec when available (CPython only), falling back to the standard library json module otherwise.

angr.utils.json_utils.**json_decode**(*data*)

**Return type**
Any

**Parameters**
**data** (bytes)

angr.utils.json_utils.**json_encode**(*obj*)

**Return type**
bytes

**Parameters**
**obj** (*Any*)

### 10.46.21 angr.utils.lazy_import

angr.utils.lazy_import.**lazy_import**(*name*)

### 10.46.22 angr.utils.library

angr.utils.library.**get_function_name**(*s*)

Get the function name from a C-style function declaration string.

**Parameters**
**s** (*str*) – A C-style function declaration string.

**Returns**
The function name.

**Return type**
str

angr.utils.library.**register_kernel_types**()

angr.utils.library.**convert_cproto_to_py**(*c_decl*)

Convert a C-style function declaration string to its corresponding SimTypes-based Python representation.

**Parameters**
**c_decl** (*str*) – The C-style function declaration string.

**Return type**
tuple[str, *SimTypeFunction*, str]

**Returns**
A tuple of the function name, the prototype, and a string representing the SimType-based Python representation.

angr.utils.library.**convert_cppproto_to_py**(*cpp_decl*, *with_param_names=False*)

Pre-process a C++-style function declaration string to its corresponding SimTypes-based Python representation.

**Parameters**

- **cpp_decl** (str) – The C++-style function declaration string.
- **with_param_names** (*bool*)

**Return type**
tuple[str | None, *SimTypeCppFunction* | None, str | None]

**Returns**
A tuple of the function name, the prototype, and a string representing the SimType-based Python representation.

angr.utils.library.**parsedcprotos2py**(*parsed_cprotos*, *fd_spots=frozenset({})*, *remove_sys_prefix=False*)

Parse a list of C function declarations and output to Python code that can be embedded into angr.procedures.definitions.

```
>>> # parse the list of glibc C prototypes and output to a file
>>> from angr.procedures.definitions import glibc
>>> with open("glibc_protos", "w") as f: f.write(cprotos2py(glibc._libc_c_decls))
```

**Parameters**
**parsed_cprotos** (list[tuple[str, *SimTypeFunction*, str]]) – A list of tuples where each tuple is (function name, parsed C function prototype, the original function declaration).

**Return type**
str

**Returns**
A Python string.

angr.utils.library.**cprotos2py**(*cprotos*, *fd_spots=frozenset({})*, *remove_sys_prefix=False*)

Parse a list of C function declarations and output to Python code that can be embedded into angr.procedures.definitions.

```
>>> # parse the list of glibc C prototypes and output to a file
>>> from angr.procedures.definitions import glibc
>>> with open("glibc_protos", "w") as f: f.write(cprotos2py(glibc._libc_c_decls))
```

**Parameters**
**cprotos** (list[str]) – A list of C prototype strings.

**Return type**
str

**Returns**
A Python string.

angr.utils.library.**get_cpp_function_name_and_metadata**(*demangled_name*)

Parse a demangled C++ declaration into a function name.

Note that the extracted name may include template instantiation, for example:

example_func<int>

**Parameters**
**demangled_name** (str) – The demangled C++ function name.

**Return type**
tuple[str, dict[str, bool]]

**Returns**
The qualified function name, excluding return type and parameters, and a dictionary with keys indicating if the function is a constructor or a destructor.

angr.utils.library.**get_cpp_function_name**(*demangled_name*)

Parse a demangled C++ declaration into a function name.

Note that the extracted name may include template instantiation, for example:

example_func<int>

**Parameters**
**demangled_name** (str) – The demangled C++ function name.

**Return type**
str

**Returns**
The qualified function name, excluding return type and parameters.

angr.utils.library.**get_rust_function_name**(*demangled_name*)

### 10.46.23 angr.utils.loader

angr.utils.loader.**is_pc**(*project*, *ins_addr*, *addr*)

Check if the given address is program counter (PC) or not. This function is for handling the case on some bizarre architectures where PC is always the currently executed instruction address plus a constant value.

**Parameters**

- **project** (*Project*) – An angr Project instance.
- **ins_addr** (int) – The address of an instruction. We calculate PC using this instruction address.
- **addr** (int) – The address to check against.

**Return type**
bool

**Returns**
True if the given instruction address is the PC, False otherwise.

angr.utils.loader.**is_in_readonly_section**(*project*, *addr*)

Check if the specified address is inside a read-only section.

**Parameters**

- **project** (*Project*) – An angr Project instance.
- **addr** (int) – The address to check.

**Return type**
bool

**Returns**
True if the given address belongs to a read-only section, False otherwise.

angr.utils.loader.**is_in_readonly_segment**(*project*, *addr*)

Check if the specified address is inside a read-only segment.

**Parameters**

- **project** (*Project*) – An angr Project instance.
- **addr** (int) – The address to check.

**Return type**
bool

**Returns**
True if the given address belongs to a read-only segment, False otherwise.

### 10.46.24 angr.utils.mp

**class** angr.utils.mp.**Closure**

Bases: NamedTuple

A pickle-able lambda; note that f, args, and kwargs must be pickleable

**f:** Callable[[...], None]

Alias for field number 0

**args:** list[Any]

Alias for field number 1

**kwargs:** dict[str, Any]

Alias for field number 2

**static __new__**(*_cls*, *f:* Callable*[..., None]*, *args: list[Any]*, *kwargs: dict[str, Any]*)

Create new instance of Closure(f, args, kwargs)

**Parameters**

- **f** (*Callable[..., None]*)
- **args** (*list[Any]*)
- **kwargs** (*dict[str, Any]*)

class angr.utils.mp.**Initializer**

Bases: object

A singleton class with global state used to initialize a multiprocessing.Process

**classmethod get**()

A wrapper around init since this class is a singleton

**Return type**
*Initializer*

**__init__**(*, *_manual=True*)

**Parameters**
**_manual** (*bool*)

**initializers:** list[*Closure*]

**register**(*f*, **args*, ***kwargs*)

A shortcut for adding Closures as initializers

**Return type**
None

**Parameters**

- **f** (*Callable[[...], None]*)
- **args** (*Any*)
- **kwargs** (*Any*)

**initialize**()

Initialize a multiprocessing.Process Set the current global initializer to the same state as this initializer, then calls each initializer

**Return type**
None

angr.utils.mp.**mp_context**()

angr.utils.mp.**protect_stdio_from_forked_children**()

Make forked worker processes safe to use from a process whose stdin/stdout are a protocol channel.

**Return type**
None

### 10.46.25 angr.utils.orderedset

**class** angr.utils.orderedset.**OrderedSet**

Bases: MutableSet, Generic

Adapted from http://code.activestate.com/recipes/576694/ Originally created by Raymond Hettinger and licensed under MIT.

**__init__**(*iterable=None*)

### 10.46.26 angr.utils.smart_cache

**class** angr.utils.smart_cache.**SmartLRUCache**

Bases: LRUCache

An LRU cache that supports an eviction callback.

**__init__**(*maxsize*, *getsizeof=None*, *evict=None*)

**Parameters**
**evict** (*Callable | None*)

### 10.46.27 angr.utils.ssa

**class** angr.utils.ssa.**AILWhitelistExprTypeWalker**

Bases: *AILBlockViewer*

Walks an AIL expression or statement and determines if it is built *only* out of a whitelisted set of expression types. has_nonwhitelisted_exprs is set to True as soon as any expression whose type is not in the whitelist is encountered.

Note that the whitelist must include the operator/container types that the walker recurses through (e.g. BinaryOp, UnaryOp), otherwise those nodes would be flagged as non-whitelisted.

**__init__**(*whitelist_expr_types*)

**Parameters**
**whitelist_expr_types** (*tuple[type, ...]*)

**class** angr.utils.ssa.**VVarUsesCollector**

Bases: *AILBlockViewer*

Collect all uses of virtual variables and their use locations in an AIL block. Skip collecting use locations if block is not specified.

**__init__**()

**vvar_and_uselocs:** dict[int, list[tuple[*VirtualVariable*, *AILCodeLocation*]]]

**vvars:** set[int]

angr.utils.ssa.**check_in_between_stmts**(*graph*, *blocks*, *defloc*, *useloc*, *predicate*)

**Parameters**

- **graph** (*DiGraph*)
- **blocks** (*dict[tuple[int, int | None], Block]*)
- **defloc** (AILCodeLocation)
- **useloc** (AILCodeLocation)

- **predicate** (*Callable*)

angr.utils.ssa.**get_tmp_deflocs**(*blocks*)

**Return type**
dict[tuple[int, int | None], dict[*Tmp*, int]]

**Parameters**
**blocks** (*Iterable[Block]*)

angr.utils.ssa.**get_tmp_uselocs**(*blocks*)

**Return type**
dict[tuple[int, int | None], dict[*Tmp*, set[tuple[*Tmp*, int]]]]

**Parameters**
**blocks** (*Iterable[Block]*)

angr.utils.ssa.**get_vvar_deflocs**(*blocks*, *phi_vvars=None*, *check_extra_defs=True*)

**Return type**
dict[int, tuple[*VirtualVariable*, *AILCodeLocation*]]

**Parameters**

- **phi_vvars** (*dict[int, set[int | None]] | None*)
- **check_extra_defs** (*bool*)

angr.utils.ssa.**get_vvar_uselocs**(*blocks*)

**Return type**
dict[int, list[tuple[*VirtualVariable*, *AILCodeLocation*]]]

angr.utils.ssa.**has_call_in_between_stmts**(*graph*, *blocks*, *defloc*, *useloc*, *skip_if_contains_vvar=None*)

**Return type**
bool

**Parameters**

- **graph** (*DiGraph*)
- **blocks** (*dict[tuple[int, int | None], Block]*)
- **defloc** (AILCodeLocation)
- **useloc** (AILCodeLocation)
- **skip_if_contains_vvar** (*int | None*)

angr.utils.ssa.**has_ite_expr**(*expr*)

**Return type**
bool

**Parameters**
**expr** (Expression)

angr.utils.ssa.**has_load_expr_in_between_stmts**(*graph*, *blocks*, *defloc*, *useloc*, *skip_if_contains_vvar=None*)

**Return type**
bool

**Parameters**

- **graph** (*DiGraph*)
- **blocks** (*dict[tuple[int, int | None], Block]*)
- **defloc** (AILCodeLocation)
- **useloc** (AILCodeLocation)
- **skip_if_contains_vvar** (*int | None*)

angr.utils.ssa.**has_store_stmt_in_between_stmts**(*graph*, *blocks*, *defloc*, *useloc*)

**Return type**
bool

**Parameters**

- **graph** (*DiGraph*)
- **blocks** (*dict[tuple[int, int | None], Block]*)
- **defloc** (AILCodeLocation)
- **useloc** (AILCodeLocation)

angr.utils.ssa.**is_const_and_vvar_assignment**(*stmt*, *walker_cached=None*)

**Return type**
bool

**Parameters**

- **stmt** (Statement)
- **walker_cached** (AILWhitelistExprTypeWalker | *None*)

angr.utils.ssa.**is_const_assignment**(*stmt*, *only_consts=False*)

**Return type**
tuple[bool, *Const* | *StackBaseOffset* | None]

**Parameters**

- **stmt** (Statement)
- **only_consts** (*bool*)

angr.utils.ssa.**is_const_vvar_load_assignment**(*stmt*, *walker_cached=None*)

**Return type**
bool

**Parameters**

- **stmt** (Statement)
- **walker_cached** (AILWhitelistExprTypeWalker | *None*)

angr.utils.ssa.**is_const_vvar_load_dirty_assignment**(*stmt*, *walker_cached=None*)

**Return type**
bool

**Parameters**

- **stmt** (Statement)
- **walker_cached** (AILWhitelistExprTypeWalker | *None*)

angr.utils.ssa.**is_phi_assignment**(*stmt*)

**Return type**
bool

**Parameters**
**stmt** (Statement)

angr.utils.ssa.**is_vvar_eliminatable**(*vvar*, *def_stmt*)

**Return type**
bool

**Parameters**

- **vvar** (VirtualVariable)
- **def_stmt** (Statement | *None*)

angr.utils.ssa.**phi_assignment_get_src**(*stmt*)

**Return type**
*Phi* | None

**Parameters**
**stmt** (Statement)

## Submodules

| | |
|---|---|
| *combined_uses_collector* | Combined walker that collects VVar uses + Tmp uses in a single pass. |
| *tmp_uses_collector* | |
| *vvar_extra_defs_collector* | |
| *vvar_uses_collector* | |

### angr.utils.ssa.combined_uses_collector

Combined walker that collects VVar uses + Tmp uses in a single pass.

SPropagator's `_analyze` historically called four separate helpers (`get_vvar_deflocs`, `get_vvar_uselocs`, `get_tmp_deflocs`, `get_tmp_uselocs`) against the same set of blocks. Each of the two `_uselocs` helpers spun up its own `AILBlockViewer` and walked every block independently.

Folding both use-collectors into a single walker halves the walk-the-block cost (one `_handle_expr` dispatch chain per visited node, not two) and is a drop-in replacement: the per-block tmp use extraction matches `TmpUsesCollector`'s shape and the cross-block vvar use accumulator matches `VVarUsesCollector`'s.

**class** angr.utils.ssa.combined_uses_collector.**VVarAndTmpUsesCollector**

Bases: *AILBlockViewer*

Collect VVar uses + Tmp uses in a single pass.

See `VVarUsesCollector` / `TmpUsesCollector` for the per-side semantics. This combined version inherits both:

- `vvar_and_uselocs` accumulates across all blocks walked (consumer extracts when done).
- `tmp_and_uselocs` is intended to be extracted per-block; `reset_tmp_uses_only()` clears just the tmp side without losing the cross-block vvar accumulator.

**__init__**()

**Return type**
None

**reset_tmp_uses_only**()

**Return type**
None

### angr.utils.ssa.tmp_uses_collector

**class** angr.utils.ssa.tmp_uses_collector.**TmpUsesCollector**

Bases: *AILBlockViewer*

Collect all uses of temporary variables and their use statement IDs in an AIL block.

**__init__**()

### angr.utils.ssa.vvar_extra_defs_collector

**class** angr.utils.ssa.vvar_extra_defs_collector.**FindExtraDefs**

Bases: *AILBlockViewer*

Simple walker that finds any extra defs

**__init__**(**args*, ***kwargs*)

### angr.utils.ssa.vvar_uses_collector

**class** angr.utils.ssa.vvar_uses_collector.**VVarUsesCollector**

Bases: *AILBlockViewer*

Collect all uses of virtual variables and their use locations in an AIL block. Skip collecting use locations if block is not specified.

**__init__**()

**vvar_and_uselocs:** dict[int, list[tuple[*VirtualVariable*, *AILCodeLocation*]]]

**vvars:** set[int]

## 10.46.28 angr.utils.strings

angr.utils.strings.**decode_utf16_string**(*data*)

Decode a UTF-16 encoded string from a bytes object in a resilient manner.

**Parameters**

- **data** (bytes) – The bytes object containing the UTF-16 encoded string.
- **errors** – The error handling scheme. Default is 'strict'. Other options include 'ignore', 'replace', etc.

**Return type**
str

**Returns**
The decoded string.

### 10.46.29 angr.utils.tagged_interval_map

**class** angr.utils.tagged_interval_map.**TaggedIntervalMap**

Bases: object

Catalogs features of intervals.

**__init__**(*nbits=0*)

**Parameters**
**nbits** (int) – Number of binning bits. Higher values reduce detail. 0 for no binning.

**property nbits: int**

**add**(*addr*, *size*, *tags*)

Add interval starting at *addr* of *size* bytes.

When binning, intervals endpoints are aligned to 2^nbits. Gaps between added intervals are populated with implicit intervals having tag value of 0. Overlapping intervals will have tag bits OR'd together.

Adjacent intervals in the interval map have unique tags. When intervals are added to the map, any adjacent stops with identical tags will be eliminated so the map retains this property.

For example: if an interval(addr=0, size=100, tags=1) is added, followed by (100, 100, 1), the resulting interval in the map will be (0, 200, 1).

**Return type**
None

**Parameters**

- **addr** (*int*)
- **size** (*int*)
- **tags** (*int*)

**irange**(*min_addr=None*, *max_addr=None*)

Iterate over intervals intersecting [min_addr, max_addr], yielding interval (addr, size, tags) tuples. Implicit gap intervals (with tags=0) are also returned.

**Parameters**

- **min_addr** (int | None) – Minimum address (inclusive) to begin iterating from. If None, iterate from start of map.
- **max_addr** (int | None) – Maximum address (inclusive) to iterate to. If None, iterate to end of map.

**Return type**
Iterator[tuple[int, int, int]]

### 10.46.30 angr.utils.timing

angr.utils.timing.**print_timing_total**()

angr.utils.timing.**timethis**(*func*)

### 10.46.31 angr.utils.types

angr.utils.types.**unpack_typeref**(*ty*)

angr.utils.types.**unpack_pointer**(*ty*, *iterative=False*)

**Return type**
*SimType* | None

**Parameters**

- **ty** (SimType)
- **iterative** (*bool*)

angr.utils.types.**unpack_pointer_and_array**(*ty*, *iterative=False*)

**Return type**
*SimType* | None

**Parameters**

- **ty** (SimType)
- **iterative** (*bool*)

angr.utils.types.**replace_pointer_pts_to**(*ty*, *old_pts_to*, *new_pts_to*)

**Return type**
*SimTypePointer* | None

**Parameters**

- **ty** (SimType)
- **old_pts_to** (SimType)
- **new_pts_to** (SimType)

angr.utils.types.**unpack_array**(*ty*)

**Return type**
*SimType* | None

angr.utils.types.**squash_array_reference**(*ty*)

angr.utils.types.**dereference_simtype**(*t*, *type_collections*, *memo=None*, *keep_missing=False*, *arch=None*)

Replace every SimTypeRef inside *t* with the real type from *type_collections*.

When arch is not None, memo holds the with_arch() copies of every dereferenced struct and union. This way we can avoid creating duplicate arch-ed copies of the same type.

**Parameters**

- **keep_missing** (bool) – Leave a SimTypeRef in place when no collection defines it, instead of raising AngrMissingTypeError.
- **t** (SimType)
- **type_collections** (*list[SimTypeCollection]*)
- **memo** (*dict[str | int, SimType] | None*)
- **arch** (*Arch | None*)

**Return type**
*SimType*

angr.utils.types.**type_collections_for_lib**(*libname*)

The type collections that prototypes of library *libname* refer to. Falls back to every loaded type collection when the library is unknown or declares no type collections.

**Return type**
list[*SimTypeCollection*]

**Parameters**
**libname** (*str | None*)

angr.utils.types.**dereference_simtype_by_lib**(*t*, *libname*)

**Return type**
*SimType*

**Parameters**

- **t** (SimType)
- **libname** (*str*)

angr.utils.types.**relink_typerefs**(*t*, *typerefs*, *_seen=None*)

Replace every TypeRef inside *t* whose name is in *typerefs* with the TypeRef object from *typerefs*, so that deserialized types share the TypeRef objects owned by a TypesStore again. The TypeRef targets themselves are not descended into; *t* is updated in place.

**Return type**
*SimType*

**Parameters**

- **t** (SimType)
- **typerefs** (*dict[str, TypeRef]*)
- **_seen** (*dict[int, SimType] | None*)

angr.utils.types.**find_type_refs**(*t*, *_seen=None*)

Collect the names of all SimTypeRef nodes reachable from *t*.

**Return type**
set[str]

**Parameters**

- **t** (SimType)
- **_seen** (*set[int] | None*)

angr.utils.types.**make_type_reference**(*t*, *memo=None*, *type_collections=None*)

Take a SimType and convert named SimStruct instances to SimTypeRefs.

**Parameters**

- **t** (*SimType*) – The SimType instance to convert.
- **type_collections** (list[*SimTypeCollection*] | None) – Only structs defined in one of these collections are converted, so that the references can be resolved later. None converts every named struct.
- **memo** (*dict[str, SimTypeRef] | None*)

**Return type**
*SimType*

**Returns**
A converted SimType instance.

### 10.46.32 angr.utils.vex

**class** angr.utils.vex.**InsAddrSeq**

Bases: Protocol

The instruction-address containers angr passes around: a plain list of addresses, or the packed InsAddrList a CFG node carries. InsAddrList.__getitem__ is unannotated and also serves slices, so the element type cannot be pinned to int here.

**__init__**(**args*, ***kwargs*)

angr.utils.vex.**get_tmp_def_stmt**(*vex_block*, *tmp_idx*)

**Return type**
int | None

**Parameters**

- **vex_block** (*IRSB*)
- **tmp_idx** (*int*)

angr.utils.vex.**block_branch_ins_addr**(*ins_addrs*, *block_addr*, *block_size*, *arch*)

Determine the address of the control-transfer instruction that ends a block.

On architectures without delay slots, this is the last instruction of the block. On delay-slot architectures (MIPS), the block ends with the delay slot and the branch precedes it; depending on the lifter, the branch and its delay slot are recorded either as two IMarks (libVEX forks before Valgrind 3.27.1, pcode) or as a single IMark spanning both instructions (Valgrind 3.27.1+), in which case the last recorded instruction address is the branch itself.

**Parameters**

- **ins_addrs** (*InsAddrSeq*) – Instruction addresses of the block (IMark start addresses).
- **block_addr** (int) – Address of the block.
- **block_size** (int | None) – Size of the block in bytes, or None when it is unknown.
- **arch** (Arch) – The architecture.

**Return type**
int | None

**Returns**
The branch instruction address, or None if ins_addrs is empty.

angr.utils.vex.**block_is_single_instruction**(*ins_addrs*, *block_addr*, *block_size*, *arch*)

Determine whether a block holds at most one instruction, i.e. it is too short to carry a branch together with its delay slot. Only libVEX (Valgrind 3.27.1+) merges a branch and its delay slot into one IMark; such an IMark spans two fixed-width instructions, so on libVEX architectures a lone IMark counts as a single instruction only if it is no longer than one instruction.

**Parameters**

- **ins_addrs** (*InsAddrSeq*) – Instruction addresses of the block (IMark start addresses).
- **block_addr** (int) – Address of the block.
- **block_size** (int) – Size of the block in bytes.
- **arch** (Arch) – The architecture.

**Return type**
bool

**Returns**
True if the block holds at most one instruction.

## 10.47 angr.vaults

**class** angr.vaults.**VaultPickler**

Bases: Pickler

**__init__**(*vault*, *file*, **args*, *assigned_objects=()*, ***kwargs*)

A persistence-aware pickler. It will check for persistence of any objects except for those with IDs in 'assigned_objects'.

**persistent_id**(*obj*)

**class** angr.vaults.**VaultUnpickler**

Bases: Unpickler

**persistent_load**(*pid*)

**class** angr.vaults.**Vault**

Bases: MutableMapping

The vault is a serializer for angr.

**keys**()

Should return the IDs stored by the vault.

**__init__**()

**is_stored**(*i*)

Checks if the provided id is already in the vault.

**load**(*oid*)

**store**(*o*)

**dumps**(*o*)

Returns a serialized string representing the object, post-deduplication.

**Parameters**
**o** – the object

**loads**(*s*)

Deserializes a string representation of the object.

**Parameters**
**s** – the string

**static close**()

**class** angr.vaults.**VaultDict**

Bases: *Vault*

A Vault that uses a dictionary for storage.

**__init__**(*d=None*)

class angr.vaults.**VaultDir**

Bases: *Vault*

A Vault that uses a directory for storage.

**__init__**(*d=None*)

class angr.vaults.**VaultShelf**

Bases: *VaultDict*

A Vault that uses a shelve.Shelf for storage.

**__init__**(*path=None*)

**close**()

class angr.vaults.**VaultDirShelf**

Bases: *VaultDict*

A Vault that uses a directory for storage, where every object is stored into a single shelve.Shelf instance. VaultDir creates a file for each object. VaultDirShelf creates only one file for a stored object and everything else it references.

**__init__**(*d=None*)

**store**(*o*)

**load**(*oid*)

CHAPTER

ELEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## a


angr, 155
angr.ail_callable, 208
angr.ailment, 208
angr.ailment.block, 214
angr.ailment.block_walker, 215
angr.ailment.constant, 216
angr.ailment.converter_common, 216
angr.ailment.converter_pcode, 217
angr.ailment.converter_vex, 217
angr.ailment.expression, 217
angr.ailment.manager, 224
angr.ailment.statement, 224
angr.ailment.tagged_object, 226
angr.ailment.utils, 227
angr.analyses, 228
angr.analyses.analysis, 275
angr.analyses.backward_slice, 277
angr.analyses.binary_optimizer, 279
angr.analyses.bindiff, 280
angr.analyses.boyscout, 283
angr.analyses.callee_cleanup_finder, 283
angr.analyses.calling_convention, 283
angr.analyses.calling_convention.calling_convention, 285
angr.analyses.calling_convention.fact_collector, 286
angr.analyses.calling_convention.utils, 289
angr.analyses.cdg, 289
angr.analyses.cfg, 290
angr.analyses.cfg.cfb, 304
angr.analyses.cfg.cfg, 306
angr.analyses.cfg.cfg_arch_options, 307
angr.analyses.cfg.cfg_base, 307
angr.analyses.cfg.cfg_emulated, 310
angr.analyses.cfg.cfg_fast, 315
angr.analyses.cfg.cfg_fast_soot, 325
angr.analyses.cfg.cfg_job_base, 326
angr.analyses.cfg.go_prologue, 326
angr.analyses.cfg.indirect_jump_resolvers, 327
angr.analyses.cfg.indirect_jump_resolvers.aarch64_macho_got, 331
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got, 331
angr.analyses.cfg.indirect_jump_resolvers.amd64_pe_iat, 332
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast, 332
angr.analyses.cfg.indirect_jump_resolvers.const_resolver, 332
angr.analyses.cfg.indirect_jump_resolvers.constant_value_m
333
angr.analyses.cfg.indirect_jump_resolvers.default_resolver
333
angr.analyses.cfg.indirect_jump_resolvers.jumptable, 333
angr.analyses.cfg.indirect_jump_resolvers.memload_resolver
338
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast, 338
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_got, 339
angr.analyses.cfg.indirect_jump_resolvers.propagator_utils
339
angr.analyses.cfg.indirect_jump_resolvers.resolver, 340
angr.analyses.cfg.indirect_jump_resolvers.syscall_resolver
340
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt, 341
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat, 341
angr.analyses.cfg.meta_structs, 341
angr.analyses.cfg.pe_msvc_eh_structs, 342
angr.analyses.cfg_slice_to_sink, 346
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink, 348
angr.analyses.cfg_slice_to_sink.graph, 350
angr.analyses.cfg_slice_to_sink.transitions, 351
angr.analyses.class_identifier, 351
angr.analyses.code_tagging, 351
angr.analyses.codecave, 352

angr.analyses.complete_calling_conventions, 353
angr.analyses.congruency_check, 354
angr.analyses.data_dep, 355
angr.analyses.data_dep.data_dependency_analysis, 358
angr.analyses.data_dep.dep_nodes, 359
angr.analyses.data_dep.sim_act_location, 361
angr.analyses.datagraph_meta, 361
angr.analyses.ddg, 361
angr.analyses.decompiler, 367
angr.analyses.decompiler.ail_simplifier, 396
angr.analyses.decompiler.ailgraph_walker, 398
angr.analyses.decompiler.block_io_finder, 398
angr.analyses.decompiler.block_similarity, 398
angr.analyses.decompiler.block_simplifier, 399
angr.analyses.decompiler.block_walkers, 401
angr.analyses.decompiler.callsite_maker, 401
angr.analyses.decompiler.ccall_rewriters, 402
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls, 402
angr.analyses.decompiler.ccall_rewriters.arm_ccalls, 402
angr.analyses.decompiler.ccall_rewriters.rewriter_base, 402
angr.analyses.decompiler.ccall_rewriters.x86_ccalls, 403
angr.analyses.decompiler.clinic, 403
angr.analyses.decompiler.condition_processor, 408
angr.analyses.decompiler.counters, 410
angr.analyses.decompiler.counters.boolean_counter, 411
angr.analyses.decompiler.counters.call_counter, 411
angr.analyses.decompiler.counters.expression_counter, 412
angr.analyses.decompiler.counters.seq_cf_structure_counter, 412
angr.analyses.decompiler.decompilation_cache, 412
angr.analyses.decompiler.decompilation_options, 413
angr.analyses.decompiler.decompiler, 414
angr.analyses.decompiler.dephication, 417
angr.analyses.decompiler.dephication.dephication_base, 418
angr.analyses.decompiler.dephication.graph_dephication, 419
angr.analyses.decompiler.dephication.graph_rewriting, 419
angr.analyses.decompiler.dephication.graph_vvar_mapping, 419
angr.analyses.decompiler.dephication.rewriting_engine, 419
angr.analyses.decompiler.dephication.seqnode_dephication, 420
angr.analyses.decompiler.dirty_rewriters, 421
angr.analyses.decompiler.dirty_rewriters.amd64_dirty, 421
angr.analyses.decompiler.dirty_rewriters.rewriter_base, 421
angr.analyses.decompiler.edits, 421
angr.analyses.decompiler.edits.cache, 435
angr.analyses.decompiler.edits.errors, 437
angr.analyses.decompiler.edits.hooks, 438
angr.analyses.decompiler.edits.ops, 442
angr.analyses.decompiler.edits.resolve, 446
angr.analyses.decompiler.edits.results, 449
angr.analyses.decompiler.empty_node_remover, 450
angr.analyses.decompiler.expression_narrower, 451
angr.analyses.decompiler.goto_manager, 452
angr.analyses.decompiler.jump_target_collector, 453
angr.analyses.decompiler.jumptable_entry_condition_rewriter, 453
angr.analyses.decompiler.known_patterns, 453
angr.analyses.decompiler.known_patterns.apply, 485
angr.analyses.decompiler.known_patterns.block_split, 486
angr.analyses.decompiler.known_patterns.containing_record, 487
angr.analyses.decompiler.known_patterns.context, 487
angr.analyses.decompiler.known_patterns.ctype_tables, 489
angr.analyses.decompiler.known_patterns.dsl, 490
angr.analyses.decompiler.known_patterns.finder, 507
angr.analyses.decompiler.known_patterns.gating, 510
angr.analyses.decompiler.known_patterns.generator, 515
angr.analyses.decompiler.known_patterns.kernel_err, 516
angr.analyses.decompiler.known_patterns.layouts, 517
angr.analyses.decompiler.known_patterns.libm_bits, 518
angr.analyses.decompiler.known_patterns.linked_list, 518
angr.analyses.decompiler.known_patterns.pattern,

519
angr.analyses.decompiler.known_patterns.posix_macros, 523
angr.analyses.decompiler.known_patterns.protobuf_hasbits, 524
angr.analyses.decompiler.known_patterns.registry, 524
angr.analyses.decompiler.known_patterns.std_string_cstr, 526
angr.analyses.decompiler.known_patterns.std_string_dtor, 526
angr.analyses.decompiler.known_patterns.std_string_internals, 527
angr.analyses.decompiler.known_patterns.std_string_length, 527
angr.analyses.decompiler.known_patterns.std_swap, 527
angr.analyses.decompiler.known_patterns.std_vector_size, 528
angr.analyses.decompiler.known_patterns.stl_accessors2, 530
angr.analyses.decompiler.known_patterns.stl_containers, 531
angr.analyses.decompiler.known_patterns.templates, 531
angr.analyses.decompiler.known_patterns.vector_claims, 533
angr.analyses.decompiler.known_patterns.vector_math, 534
angr.analyses.decompiler.known_patterns.wdk_shared_data, 534
angr.analyses.decompiler.label_collector, 535
angr.analyses.decompiler.node_replacer, 535
angr.analyses.decompiler.notes, 535
angr.analyses.decompiler.notes.decompilation_notes, 538
angr.analyses.decompiler.notes.deobfuscated_strings, 539
angr.analyses.decompiler.optimization_pass_registry, 540
angr.analyses.decompiler.optimization_passes, 540
angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier, 559
angr.analyses.decompiler.optimization_passes.call_stmt_rewriter, 560
angr.analyses.decompiler.optimization_passes.code_motion, 560
angr.analyses.decompiler.optimization_passes.conditional_constraints_propagation, 562
angr.analyses.decompiler.optimization_passes.const_derefs, 562
angr.analyses.decompiler.optimization_passes.const_prop_reverter, 563
angr.analyses.decompiler.optimization_passes.cross_jump_reverter, 564
angr.analyses.decompiler.optimization_passes.deadblock_remover, 565
angr.analyses.decompiler.optimization_passes.determine_loop_max_iterations, 565
angr.analyses.decompiler.optimization_passes.div_simplifier, 565
angr.analyses.decompiler.optimization_passes.duplication_reverter, 566
angr.analyses.decompiler.optimization_passes.duplication_reverter.block_similarity, 568
angr.analyses.decompiler.optimization_passes.duplication_reverter.duplication_finder, 570
angr.analyses.decompiler.optimization_passes.duplication_reverter.duplication_reverter, 571
angr.analyses.decompiler.optimization_passes.duplication_reverter.structure_matcher, 572
angr.analyses.decompiler.optimization_passes.duplication_reverter.utils, 572
angr.analyses.decompiler.optimization_passes.eager_std_string_concatenation, 573
angr.analyses.decompiler.optimization_passes.eager_std_string_concatenation_windows, 574
angr.analyses.decompiler.optimization_passes.engine_base, 574
angr.analyses.decompiler.optimization_passes.expr_op_swapper, 575
angr.analyses.decompiler.optimization_passes.flip_boolean_cmp, 576
angr.analyses.decompiler.optimization_passes.inlined_memcpy, 577
angr.analyses.decompiler.optimization_passes.inlined_memset, 577
angr.analyses.decompiler.optimization_passes.inlined_strcpy, 578
angr.analyses.decompiler.optimization_passes.inlined_strlen, 579
angr.analyses.decompiler.optimization_passes.inlined_strncpy, 581
angr.analyses.decompiler.optimization_passes.inlined_wcscpy, 581
angr.analyses.decompiler.optimization_passes.ite_expr_converter, 582
angr.analyses.decompiler.optimization_passes.ite_region_converter, 583
angr.analyses.decompiler.optimization_passes.known_pattern_identifier, 583
angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier, 584
angr.analyses.decompiler.optimization_passes.mips_gp_setting, 585
angr.analyses.decompiler.optimization_passes.mod_simplifier, 586

angr.analyses.decompiler.optimization_passes.optimization_pass, 586
angr.analyses.decompiler.optimization_passes.peephole_simplifier, 589
angr.analyses.decompiler.optimization_passes.register_saves_decompiler_pre, 590
angr.analyses.decompiler.optimization_passes.register_saves_decompiler_pre_hole, 590
angr.analyses.decompiler.optimization_passes.regrader_saves_decompiler, 591
angr.analyses.decompiler.optimization_passes.regrader_saves_decompiler, 591
angr.analyses.decompiler.optimization_passes.return_duplicator, 592
angr.analyses.decompiler.optimization_passes.return_duplicator_base, 592
angr.analyses.decompiler.optimization_passes.return_duplicator_high, 593
angr.analyses.decompiler.optimization_passes.return_duplicator_low, 594
angr.analyses.decompiler.optimization_passes.stack_analyses_decompiler, 594
angr.analyses.decompiler.optimization_passes.stack_analyses_decompiler, 596
angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator, 597
angr.analyses.decompiler.optimization_passes.switch_analyses_decompiler_pre, 597
angr.analyses.decompiler.optimization_passes.tagger_analyses, 598
angr.analyses.decompiler.optimization_passes.win_stack_analyses_decompiler, 598
angr.analyses.decompiler.optimization_passes.x86_gcc_analyses_decompiler, 598
angr.analyses.decompiler.peephole_optimizations, 600
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler_peephole_opt, 600
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler_pre, 600
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler, 601
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler, 601
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler, 601
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler, 601
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler, 602
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler, 602
angr.analyses.decompiler.peephole_optimizations.a_and_cons_decompiler, 605
angr.analyses.decompiler.peephole_optimizations.basepointe, 605
angr.analyses.decompiler.peephole_optimizations.bitwise_in, 605
angr.analyses.decompiler.peephole_optimizations.bitwise_or, 605
angr.analyses.decompiler.peephole_optimizations.bool_expr_, 606
angr.analyses.decompiler.peephole_optimizations.bswap, 606
angr.analyses.decompiler.peephole_optimizations.bt_rflags_, 606
angr.analyses.decompiler.peephole_optimizations.cas_intrin, 607
angr.analyses.decompiler.peephole_optimizations.cmp_masked, 608
angr.analyses.decompiler.peephole_optimizations.cmp_sub_co, 608
angr.analyses.decompiler.peephole_optimizations.cmpord_rew, 609
angr.analyses.decompiler.peephole_optimizations.coalesce_a, 609
angr.analyses.decompiler.peephole_optimizations.coalesce_s, 609
angr.analyses.decompiler.peephole_optimizations.concat_sim, 610
angr.analyses.decompiler.peephole_optimizations.constant_d, 610
angr.analyses.decompiler.peephole_optimizations.conv_a_sub, 610
angr.analyses.decompiler.peephole_optimizations.conv_shl_s, 610
angr.analyses.decompiler.peephole_optimizations.eager_eval, 611
angr.analyses.decompiler.peephole_optimizations.evaluate_c, 611
angr.analyses.decompiler.peephole_optimizations.extended_b, 612
angr.analyses.decompiler.peephole_optimizations.inlined_st, 612
angr.analyses.decompiler.peephole_optimizations.inlined_st, 612
angr.analyses.decompiler.peephole_optimizations.invert_neg, 612
angr.analyses.decompiler.peephole_optimizations.known_patt, 612
angr.analyses.decompiler.peephole_optimizations.modulo_sim, 613
angr.analyses.decompiler.peephole_optimizations.one_sub_bo, 613
angr.analyses.decompiler.peephole_optimizations.optimized_, 613
angr.analyses.decompiler.peephole_optimizations.remove_cas, 614

angr.analyses.decompiler.peephole_optimizations.remove_const_cmpi, 614
angr.analyses.decompiler.peephole_optimizations.remove_cxx_decompiler_calls, 614
angr.analyses.decompiler.peephole_optimizations.remove_empty_if_body, 614
angr.analyses.decompiler.peephole_optimizations.remove_noop_conversions, 615
angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks, 615
angr.analyses.decompiler.peephole_optimizations.remove_redundant_conversions, 615
angr.analyses.decompiler.peephole_optimizations.remove_redundant_derefs, 616
angr.analyses.decompiler.peephole_optimizations.remove_redundant_insert, 616
angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_branch, 616
angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_comparisons, 616
angr.analyses.decompiler.peephole_optimizations.remove_redundant_nots, 617
angr.analyses.decompiler.peephole_optimizations.remove_redundant_reinterprets, 617
angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts, 617
angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts_around_comparators, 618
angr.analyses.decompiler.peephole_optimizations.rewrite_bit_extractions, 618
angr.analyses.decompiler.peephole_optimizations.rewrite_conv_mul, 618
angr.analyses.decompiler.peephole_optimizations.rewrite_cxx_operator_calls, 619
angr.analyses.decompiler.peephole_optimizations.rewrite_mips_gp_loads, 619
angr.analyses.decompiler.peephole_optimizations.rol_ror, 619
angr.analyses.decompiler.peephole_optimizations.sar_to_signed_div, 620
angr.analyses.decompiler.peephole_optimizations.shl_to_mul, 620
angr.analyses.decompiler.peephole_optimizations.simplify_ec_decompiler_locks, 620
angr.analyses.decompiler.peephole_optimizations.simplify_condmoiber_expr, 620
angr.analyses.decompiler.peephole_optimizations.simplify_xor, 621
angr.analyses.decompiler.peephole_optimizations.tridy_stack_add, 621
angr.analyses.decompiler.peephole_optimizations.utils, 621
angr.analyses.decompiler.presets, 621
angr.analyses.decompiler.presets.basic, 622
angr.analyses.decompiler.presets.fast, 622
angr.analyses.decompiler.presets.full, 622
angr.analyses.decompiler.presets.malware, 622
angr.analyses.decompiler.presets.preset, 622
angr.analyses.decompiler.redundant_label_remover, 623
angr.analyses.decompiler.region_identifier, 623
angr.analyses.decompiler.region_overlay, 623
angr.analyses.decompiler.region_simplifiers, 633
angr.analyses.decompiler.region_simplifiers.cascading_cond 633
angr.analyses.decompiler.region_simplifiers.cascading_ifs, 633
angr.analyses.decompiler.region_simplifiers.expr_folding, 634
angr.analyses.decompiler.region_simplifiers.goto, 637
angr.analyses.decompiler.region_simplifiers.if_, 638
angr.analyses.decompiler.region_simplifiers.ifelse, 638
angr.analyses.decompiler.region_simplifiers.loop, 638
angr.analyses.decompiler.region_simplifiers.node_address_f 638
angr.analyses.decompiler.region_simplifiers.region_simplif 638
angr.analyses.decompiler.region_simplifiers.switch_cluster 639
angr.analyses.decompiler.region_simplifiers.switch_expr_si 641
angr.analyses.decompiler.region_walker, 642
angr.analyses.decompiler.return_maker, 642
angr.analyses.decompiler.semantic_naming, 642
angr.analyses.decompiler.semantic_naming.array_index_namin 647
angr.analyses.decompiler.semantic_naming.boolean_naming, 647
angr.analyses.decompiler.semantic_naming.call_result_namin 648
angr.analyses.decompiler.semantic_naming.naming_base, 648
angr.analyses.decompiler.semantic_naming.orchestrator, 650
angr.analyses.decompiler.semantic_naming.pointer_naming, 651
angr.analyses.decompiler.semantic_naming.region_loop_count 651
angr.analyses.decompiler.semantic_naming.size_naming, 652
angr.analyses.decompiler.seq_to_blocks, 652
angr.analyses.decompiler.sequence_walker, 652

angr.analyses.decompiler.ssailification, 652
angr.analyses.decompiler.ssailification.consts, 653
angr.analyses.decompiler.ssailification.rewriting, 653
angr.analyses.decompiler.ssailification.rewriting_engine, 654
angr.analyses.decompiler.ssailification.rewriting_state, 654
angr.analyses.decompiler.ssailification.ssailification, 655
angr.analyses.decompiler.ssailification.traversal, 655
angr.analyses.decompiler.ssailification.traversal_engine, 655
angr.analyses.decompiler.ssailification.traversal_state, 658
angr.analyses.decompiler.stack_item, 659
angr.analyses.decompiler.stackarg_offset_manager, 660
angr.analyses.decompiler.stl_field_accessors, 661
angr.analyses.decompiler.structured_codegen, 661
angr.analyses.decompiler.structured_codegen.base, 668
angr.analyses.decompiler.structured_codegen.c, 670
angr.analyses.decompiler.structured_codegen.c_generic, 688
angr.analyses.decompiler.structured_codegen.dummy, 691
angr.analyses.decompiler.structured_codegen.dwarf_import, 691
angr.analyses.decompiler.structured_codegen.rust, 691
angr.analyses.decompiler.structurer_nodes, 710
angr.analyses.decompiler.structuring, 714
angr.analyses.decompiler.structuring.dream, 716
angr.analyses.decompiler.structuring.phoenix, 717
angr.analyses.decompiler.structuring.recursive_structurer, 718
angr.analyses.decompiler.structuring.sailr, 718
angr.analyses.decompiler.structuring.structurer_base, 719
angr.analyses.decompiler.utils, 720
angr.analyses.decompiler.variable_map, 727
angr.analyses.deobfuscator, 731
angr.analyses.deobfuscator.api_obf_finder, 734
angr.analyses.deobfuscator.api_obf_peephole_optimizer, 735
angr.analyses.deobfuscator.api_obf_type2_finder, 736
angr.analyses.deobfuscator.data_transformation_embedder, 737
angr.analyses.deobfuscator.hash_lookup_api_deobfuscator, 737
angr.analyses.deobfuscator.irsb_reg_collector, 737
angr.analyses.deobfuscator.scope_ops_analyzer, 738
angr.analyses.deobfuscator.string_obf_finder, 738
angr.analyses.deobfuscator.string_obf_opt_passes, 739
angr.analyses.deobfuscator.string_obf_peephole_optimizer, 739
angr.analyses.disassembly, 739
angr.analyses.disassembly_utils, 743
angr.analyses.dominance_frontier, 743
angr.analyses.fcp, 744
angr.analyses.fcp.fcp, 744
angr.analyses.find_objects_static, 746
angr.analyses.flirt, 747
angr.analyses.flirt.consts, 751
angr.analyses.flirt.flirt, 755
angr.analyses.flirt.flirt_function, 756
angr.analyses.flirt.flirt_matcher, 756
angr.analyses.flirt.flirt_module, 757
angr.analyses.flirt.flirt_node, 757
angr.analyses.flirt.flirt_sig, 758
angr.analyses.flirt.flirt_utils, 761
angr.analyses.forward_analysis, 761
angr.analyses.forward_analysis.forward_analysis, 763
angr.analyses.forward_analysis.job_info, 764
angr.analyses.forward_analysis.visitors, 765
angr.analyses.forward_analysis.visitors.call_graph, 766
angr.analyses.forward_analysis.visitors.function_graph, 766
angr.analyses.forward_analysis.visitors.graph, 766
angr.analyses.forward_analysis.visitors.loop, 768
angr.analyses.forward_analysis.visitors.single_node_graph, 769
angr.analyses.identifier, 769
angr.analyses.identifier.custom_callable, 770
angr.analyses.identifier.errors, 770
angr.analyses.identifier.func, 771
angr.analyses.identifier.functions, 771
angr.analyses.identifier.functions.atoi, 772

angr.analyses.identifier.functions.based_atoi, 772
angr.analyses.identifier.functions.fdprintf, 772
angr.analyses.identifier.functions.free, 773
angr.analyses.identifier.functions.int2str, 773
angr.analyses.identifier.functions.malloc, 774
angr.analyses.identifier.functions.memcmp, 775
angr.analyses.identifier.functions.memcpy, 775
angr.analyses.identifier.functions.memset, 776
angr.analyses.identifier.functions.printf, 776
angr.analyses.identifier.functions.recv_until, 777
angr.analyses.identifier.functions.skip_calloc, 778
angr.analyses.identifier.functions.skip_realloc, 778
angr.analyses.identifier.functions.skip_recv_n, 778
angr.analyses.identifier.functions.snprintf, 779
angr.analyses.identifier.functions.sprintf, 780
angr.analyses.identifier.functions.strcasecmp, 780
angr.analyses.identifier.functions.strcmp, 781
angr.analyses.identifier.functions.strcpy, 782
angr.analyses.identifier.functions.strlen, 782
angr.analyses.identifier.functions.strncmp, 783
angr.analyses.identifier.functions.strncpy, 784
angr.analyses.identifier.functions.strtol, 784
angr.analyses.identifier.identify, 784
angr.analyses.identifier.runner, 785
angr.analyses.init_finder, 786
angr.analyses.language_detector, 786
angr.analyses.loop_analysis, 787
angr.analyses.loop_analysis.loop_analysis, 787
angr.analyses.loop_unroller, 790
angr.analyses.loop_unroller.loop_unroller, 791
angr.analyses.loopfinder, 791
angr.analyses.outliner, 791
angr.analyses.outliner.outliner, 792
angr.analyses.patchfinder, 792
angr.analyses.pathfinder, 794
angr.analyses.propagator, 796
angr.analyses.propagator.engine_base, 796
angr.analyses.propagator.engine_vex, 797
angr.analyses.propagator.propagator, 797
angr.analyses.propagator.top_checker_mixin, 797
angr.analyses.propagator.values, 798
angr.analyses.propagator.vex_vars, 798
angr.analyses.proximity_graph, 798
angr.analyses.purity, 800
angr.analyses.purity.analysis, 801
angr.analyses.purity.engine, 801
angr.analyses.reaching_definitions, 804
angr.analyses.reaching_definitions.call_trace, 834
angr.analyses.reaching_definitions.dep_graph, 835
angr.analyses.reaching_definitions.engine_ail, 838
angr.analyses.reaching_definitions.engine_vex, 838
angr.analyses.reaching_definitions.external_codeloc, 838
angr.analyses.reaching_definitions.function_handler, 838
angr.analyses.reaching_definitions.function_handler_librar 845
angr.analyses.reaching_definitions.function_handler_librar 846
angr.analyses.reaching_definitions.function_handler_librar 850
angr.analyses.reaching_definitions.function_handler_librar 852
angr.analyses.reaching_definitions.function_handler_librar 855
angr.analyses.reaching_definitions.heap_allocator, 857
angr.analyses.reaching_definitions.rd_initializer, 857
angr.analyses.reaching_definitions.rd_state, 859
angr.analyses.reaching_definitions.reaching_definitions, 867
angr.analyses.reaching_definitions.subject, 869
angr.analyses.reassembler, 870
angr.analyses.s_liveness, 878
angr.analyses.s_propagator, 879
angr.analyses.s_reaching_definitions, 880
angr.analyses.s_reaching_definitions.s_rda_model,

885
angr.analyses.s_reaching_definitions.s_rda_view, 888
angr.analyses.s_reaching_definitions.s_reaching_definitions, 891
angr.analyses.smc, 891
angr.analyses.soot_class_hierarchy, 893
angr.analyses.stack_pointer_tracker, 893
angr.analyses.static_hooker, 896
angr.analyses.typehoon, 896
angr.analyses.typehoon.dfa, 898
angr.analyses.typehoon.simple_solver, 899
angr.analyses.typehoon.translator, 905
angr.analyses.typehoon.typeconsts, 906
angr.analyses.typehoon.typehoon, 913
angr.analyses.typehoon.typevars, 914
angr.analyses.typehoon.variance, 920
angr.analyses.unpacker, 921
angr.analyses.unpacker.obfuscation_detector, 921
angr.analyses.unpacker.packing_detector, 922
angr.analyses.variable_recovery, 922
angr.analyses.variable_recovery.annotations, 923
angr.analyses.variable_recovery.engine_ail, 923
angr.analyses.variable_recovery.engine_base, 924
angr.analyses.variable_recovery.engine_vex, 925
angr.analyses.variable_recovery.irsb_scanner, 925
angr.analyses.variable_recovery.variable_recovery, 925
angr.analyses.variable_recovery.variable_recovery_base, 926
angr.analyses.variable_recovery.variable_recovery_fast, 930
angr.analyses.veritesting, 930
angr.analyses.vfg, 932
angr.analyses.vsa_ddg, 936
angr.analyses.vtable, 937
angr.analyses.xrefs, 937
angr.angrdb, 938
angr.angrdb.db, 940
angr.angrdb.models, 942
angr.angrdb.serializers, 948
angr.angrdb.serializers.bookmarks, 949
angr.angrdb.serializers.callgraph, 950
angr.angrdb.serializers.cfg_model, 951
angr.angrdb.serializers.comments, 951
angr.angrdb.serializers.funcs, 951
angr.angrdb.serializers.kb, 952
angr.angrdb.serializers.labels, 952
angr.angrdb.serializers.loader, 953
angr.angrdb.serializers.structured_code, 954
angr.angrdb.serializers.variables, 955
angr.angrdb.serializers.xrefs, 956
angr.annocfg, 957
angr.blade, 958
angr.block, 959
angr.callable, 961
angr.calling_conventions, 962
angr.code_location, 985
angr.codenode, 987
angr.concretization_strategies, 988
angr.concretization_strategies.any, 990
angr.concretization_strategies.any_named, 990
angr.concretization_strategies.base, 990
angr.concretization_strategies.controlled_data, 991
angr.concretization_strategies.eval, 991
angr.concretization_strategies.logging, 991
angr.concretization_strategies.max, 991
angr.concretization_strategies.nonzero, 992
angr.concretization_strategies.nonzero_range, 992
angr.concretization_strategies.norepeats, 992
angr.concretization_strategies.norepeats_range, 992
angr.concretization_strategies.range, 992
angr.concretization_strategies.signed_add, 992
angr.concretization_strategies.single, 992
angr.concretization_strategies.solutions, 992
angr.concretization_strategies.unlimited_range, 993
angr.distributed, 993
angr.distributed.server, 994
angr.distributed.worker, 994
angr.emulator, 995
angr.engines, 996
angr.engines.ail, 1001
angr.engines.ail.callstack, 1003
angr.engines.ail.engine_light, 1003
angr.engines.ail.engine_successors, 1004
angr.engines.ail.setup, 1004
angr.engines.engine, 1004
angr.engines.failure, 1005
angr.engines.hook, 1005
angr.engines.icicle, 1005
angr.engines.light, 1005
angr.engines.light.data, 1009
angr.engines.light.engine, 1010
angr.engines.pcode, 1014
angr.engines.pcode.behavior, 1014
angr.engines.pcode.cc, 1034
angr.engines.pcode.emulate, 1036

angr.engines.pcode.engine, 1036
angr.engines.pcode.lifter, 1037
angr.engines.procedure, 1046
angr.engines.soot, 1046
angr.engines.soot.engine, 1047
angr.engines.soot.exceptions, 1047
angr.engines.soot.expressions, 1047
angr.engines.soot.expressions.arrayref, 1050
angr.engines.soot.expressions.base, 1050
angr.engines.soot.expressions.binop, 1050
angr.engines.soot.expressions.cast, 1050
angr.engines.soot.expressions.condition, 1050
angr.engines.soot.expressions.constants, 1050
angr.engines.soot.expressions.instancefieldref, 1051
angr.engines.soot.expressions.instanceOf, 1051
angr.engines.soot.expressions.invoke, 1051
angr.engines.soot.expressions.length, 1051
angr.engines.soot.expressions.local, 1051
angr.engines.soot.expressions.new, 1052
angr.engines.soot.expressions.newArray, 1052
angr.engines.soot.expressions.newMultiArray, 1052
angr.engines.soot.expressions.paramref, 1052
angr.engines.soot.expressions.phi, 1052
angr.engines.soot.expressions.staticfieldref, 1052
angr.engines.soot.expressions.thisref, 1052
angr.engines.soot.expressions.unsupported, 1052
angr.engines.soot.field_dispatcher, 1052
angr.engines.soot.method_dispatcher, 1053
angr.engines.soot.statements, 1053
angr.engines.soot.statements.assign, 1054
angr.engines.soot.statements.base, 1054
angr.engines.soot.statements.goto, 1054
angr.engines.soot.statements.identity, 1054
angr.engines.soot.statements.if_, 1054
angr.engines.soot.statements.invoke, 1054
angr.engines.soot.statements.return_, 1054
angr.engines.soot.statements.switch, 1055
angr.engines.soot.statements.throw, 1055
angr.engines.soot.values, 1055
angr.engines.soot.values.arrayref, 1059
angr.engines.soot.values.base, 1060
angr.engines.soot.values.constants, 1060
angr.engines.soot.values.instancefieldref, 1060
angr.engines.soot.values.local, 1060
angr.engines.soot.values.paramref, 1061
angr.engines.soot.values.staticfieldref, 1061
angr.engines.soot.values.strref, 1061
angr.engines.soot.values.thisref, 1062
angr.engines.successors, 1063
angr.engines.syscall, 1065
angr.engines.unicorn, 1065
angr.engines.vex, 1065
angr.engines.vex.claripy, 1068
angr.engines.vex.claripy.ccall, 1068
angr.engines.vex.claripy.datalayer, 1073
angr.engines.vex.claripy.irop, 1073
angr.engines.vex.heavy, 1074
angr.engines.vex.heavy.actions, 1075
angr.engines.vex.heavy.concretizers, 1075
angr.engines.vex.heavy.dirty, 1076
angr.engines.vex.heavy.heavy, 1078
angr.engines.vex.heavy.inspect, 1078
angr.engines.vex.heavy.resilience, 1079
angr.engines.vex.heavy.super_fastpath, 1079
angr.engines.vex.lifter, 1079
angr.engines.vex.light, 1080
angr.engines.vex.light.light, 1080
angr.engines.vex.light.resilience, 1080
angr.engines.vex.light.slicing, 1081
angr.errors, 1081
angr.exploration_techniques, 1087
angr.exploration_techniques.base, 1097
angr.exploration_techniques.bucketizer, 1098
angr.exploration_techniques.common, 1098
angr.exploration_techniques.dfs, 1099
angr.exploration_techniques.director, 1099
angr.exploration_techniques.driller_core, 1101
angr.exploration_techniques.explorer, 1102
angr.exploration_techniques.lengthlimiter, 1102
angr.exploration_techniques.local_loop_seer, 1102
angr.exploration_techniques.loop_seer, 1102
angr.exploration_techniques.manual_mergepoint, 1103
angr.exploration_techniques.memory_watcher, 1103
angr.exploration_techniques.oppologist, 1103
angr.exploration_techniques.slicecutor, 1104
angr.exploration_techniques.spiller, 1104
angr.exploration_techniques.spiller_db, 1105
angr.exploration_techniques.stochastic, 1106
angr.exploration_techniques.stub_stasher, 1106
angr.exploration_techniques.suggestions, 1106
angr.exploration_techniques.tech_builder, 1106
angr.exploration_techniques.threading, 1107
angr.exploration_techniques.timeout, 1107
angr.exploration_techniques.tracer, 1107
angr.exploration_techniques.unique, 1109

angr.exploration_techniques.veritesting, 1109
angr.factory, 1109
angr.flirt, 1113
angr.flirt.build_sig, 1114
angr.keyed_region, 1115
angr.knowledge_base, 1118
angr.knowledge_plugins, 1120
angr.knowledge_plugins.bookmarks, 1149
angr.knowledge_plugins.callsite_prototypes, 1150
angr.knowledge_plugins.cfg, 1152
angr.knowledge_plugins.cfg.block_id, 1165
angr.knowledge_plugins.cfg.cfg_manager, 1165
angr.knowledge_plugins.cfg.cfg_model, 1166
angr.knowledge_plugins.cfg.cfg_node, 1173
angr.knowledge_plugins.cfg.indirect_jump, 1175
angr.knowledge_plugins.cfg.memory_data, 1177
angr.knowledge_plugins.cfg.spilling_cfg, 1179
angr.knowledge_plugins.cfg.spilling_digraph, 1187
angr.knowledge_plugins.cfg.types, 1189
angr.knowledge_plugins.comments, 1189
angr.knowledge_plugins.custom_strings, 1191
angr.knowledge_plugins.data, 1191
angr.knowledge_plugins.debug_variables, 1192
angr.knowledge_plugins.functions, 1194
angr.knowledge_plugins.functions.function, 1209
angr.knowledge_plugins.functions.function_manager, 1218
angr.knowledge_plugins.functions.function_parser, 1227
angr.knowledge_plugins.functions.soot_function, 1228
angr.knowledge_plugins.indirect_jumps, 1228
angr.knowledge_plugins.key_definitions, 1228
angr.knowledge_plugins.key_definitions.atoms, 1241
angr.knowledge_plugins.key_definitions.constants, 1245
angr.knowledge_plugins.key_definitions.definition, 1245
angr.knowledge_plugins.key_definitions.environment, 1247
angr.knowledge_plugins.key_definitions.heap_address, 1248
angr.knowledge_plugins.key_definitions.key_definition_manager, 1248
angr.knowledge_plugins.key_definitions.live_definitions, 1249
angr.knowledge_plugins.key_definitions.liveness, 1257
angr.knowledge_plugins.key_definitions.rd_model, 1259
angr.knowledge_plugins.key_definitions.tag, 1261
angr.knowledge_plugins.key_definitions.undefined, 1262
angr.knowledge_plugins.key_definitions.unknown_size, 1262
angr.knowledge_plugins.key_definitions.uses, 1262
angr.knowledge_plugins.labels, 1264
angr.knowledge_plugins.obfuscations, 1264
angr.knowledge_plugins.patches, 1264
angr.knowledge_plugins.plugin, 1266
angr.knowledge_plugins.propagations, 1266
angr.knowledge_plugins.propagations.prop_value, 1268
angr.knowledge_plugins.propagations.propagation_manager, 1270
angr.knowledge_plugins.propagations.propagation_model, 1271
angr.knowledge_plugins.propagations.states, 1272
angr.knowledge_plugins.rtdb, 1276
angr.knowledge_plugins.rtdb.rtdb, 1277
angr.knowledge_plugins.structured_code, 1278
angr.knowledge_plugins.types, 1280
angr.knowledge_plugins.variables, 1280
angr.knowledge_plugins.variables.spilling_vardict, 1282
angr.knowledge_plugins.variables.variable_access, 1282
angr.knowledge_plugins.variables.variable_manager, 1283
angr.knowledge_plugins.xrefs, 1290
angr.knowledge_plugins.xrefs.xref, 1291
angr.knowledge_plugins.xrefs.xref_manager, 1292
angr.knowledge_plugins.xrefs.xref_types, 1293
angr.llm_client, 1293
angr.llm_models, 1294
angr.mcp, 1295
angr.mcp.edit_tools, 1299
angr.mcp.errors, 1301
angr.mcp.serializers, 1301
angr.mcp.server, 1303
angr.mcp.session, 1307
angr.misc, 1309
angr.misc.ansi, 1314
angr.misc.autoimport, 1314
angr.misc.bug_report, 1315
angr.misc.hookset, 1315
angr.misc.loggers, 1316
angr.misc.picklable_lock, 1317
angr.misc.plugins, 1317

angr.misc.telemetry, 1320
angr.misc.testing, 1322
angr.misc.ux, 1322
angr.procedures, 1322
angr.procedures.advapi32, 1323
angr.procedures.cgc, 1323
angr.procedures.cgc.allocate, 1323
angr.procedures.cgc.deallocate, 1323
angr.procedures.cgc.fdwait, 1323
angr.procedures.cgc.random, 1323
angr.procedures.cgc.receive, 1324
angr.procedures.cgc.transmit, 1324
angr.procedures.definitions, 1324
angr.procedures.definitions.cgc, 1333
angr.procedures.definitions.gnulib, 1333
angr.procedures.definitions.libstdcpp, 1333
angr.procedures.definitions.linux_kernel, 1333
angr.procedures.definitions.linux_loader, 1333
angr.procedures.definitions.macho_libsystem, 1333
angr.procedures.definitions.msvcr, 1333
angr.procedures.definitions.parse_glibc, 1333
angr.procedures.definitions.parse_syscalls_from_local_system, 1334
angr.procedures.definitions.parse_win32json, 1334
angr.procedures.definitions.types_stl, 1334
angr.procedures.glibc, 1336
angr.procedures.glibc.dynamic_loading, 1336
angr.procedures.glibc.scanf, 1336
angr.procedures.glibc.sscanf, 1336
angr.procedures.gnulib, 1336
angr.procedures.gnulib.xalloc_die, 1337
angr.procedures.gnulib.xstrtol_fatal, 1337
angr.procedures.java, 1337
angr.procedures.java.unconstrained, 1337
angr.procedures.java_io, 1337
angr.procedures.java_io.read, 1337
angr.procedures.java_io.write, 1338
angr.procedures.java_jni, 1338
angr.procedures.java_jni.array_operations, 1338
angr.procedures.java_jni.class_and_interface_operations, 1340
angr.procedures.java_jni.field_access, 1340
angr.procedures.java_jni.global_and_local_refs, 1341
angr.procedures.java_jni.method_calls, 1342
angr.procedures.java_jni.not_implemented, 1346
angr.procedures.java_jni.object_operations, 1347
angr.procedures.java_jni.string_operations, 1347
angr.procedures.java_jni.version_information, 1347
angr.procedures.java_lang, 1348
angr.procedures.java_lang.character, 1348
angr.procedures.java_lang.double, 1348
angr.procedures.java_lang.exit, 1348
angr.procedures.java_lang.getsimplename, 1348
angr.procedures.java_lang.integer, 1348
angr.procedures.java_lang.load_library, 1349
angr.procedures.java_lang.math, 1349
angr.procedures.java_lang.string, 1349
angr.procedures.java_lang.stringbuilder, 1349
angr.procedures.java_lang.system, 1349
angr.procedures.java_util, 1349
angr.procedures.java_util.collection, 1350
angr.procedures.java_util.iterator, 1350
angr.procedures.java_util.list, 1350
angr.procedures.java_util.map, 1350
angr.procedures.java_util.random, 1350
angr.procedures.java_util.scanner_nextline, 1351
angr.procedures.libc, 1351
angr.procedures.libc.abort, 1353
angr.procedures.libc.access, 1353
angr.procedures.libc.atoi, 1353
angr.procedures.libc.atol, 1353
angr.procedures.libc.calloc, 1353
angr.procedures.libc.closelog, 1353
angr.procedures.libc.err, 1353
angr.procedures.libc.error, 1353
angr.procedures.libc.exit, 1353
angr.procedures.libc.fclose, 1354
angr.procedures.libc.feof, 1354
angr.procedures.libc.fflush, 1354
angr.procedures.libc.fgetc, 1354
angr.procedures.libc.fgets, 1354
angr.procedures.libc.fopen, 1354
angr.procedures.libc.fprintf, 1354
angr.procedures.libc.fputc, 1355
angr.procedures.libc.fputs, 1355
angr.procedures.libc.fread, 1355
angr.procedures.libc.free, 1355
angr.procedures.libc.fscanf, 1355
angr.procedures.libc.fseek, 1355
angr.procedures.libc.ftell, 1355
angr.procedures.libc.fwrite, 1356
angr.procedures.libc.getchar, 1356
angr.procedures.libc.getdelim, 1356
angr.procedures.libc.getegid, 1356
angr.procedures.libc.geteuid, 1356
angr.procedures.libc.getgid, 1356
angr.procedures.libc.gets, 1356

angr.procedures.libc.getuid, 1356
angr.procedures.libc.isalnum, 1356
angr.procedures.libc.isalpha, 1356
angr.procedures.libc.isascii, 1357
angr.procedures.libc.isblank, 1357
angr.procedures.libc.iscntrl, 1357
angr.procedures.libc.isdigit, 1357
angr.procedures.libc.isgraph, 1357
angr.procedures.libc.islower, 1357
angr.procedures.libc.isprint, 1357
angr.procedures.libc.ispunct, 1357
angr.procedures.libc.isspace, 1357
angr.procedures.libc.isupper, 1357
angr.procedures.libc.isxdigit, 1357
angr.procedures.libc.malloc, 1358
angr.procedures.libc.memccpy, 1358
angr.procedures.libc.memchr, 1358
angr.procedures.libc.memcmp, 1358
angr.procedures.libc.memcpy, 1358
angr.procedures.libc.memmove, 1358
angr.procedures.libc.mempcpy, 1358
angr.procedures.libc.memset, 1358
angr.procedures.libc.openlog, 1358
angr.procedures.libc.perror, 1358
angr.procedures.libc.printf, 1358
angr.procedures.libc.putchar, 1359
angr.procedures.libc.puts, 1359
angr.procedures.libc.rand, 1359
angr.procedures.libc.realloc, 1359
angr.procedures.libc.rewind, 1359
angr.procedures.libc.scanf, 1359
angr.procedures.libc.setbuf, 1359
angr.procedures.libc.setvbuf, 1359
angr.procedures.libc.snprintf, 1359
angr.procedures.libc.sprintf, 1359
angr.procedures.libc.srand, 1359
angr.procedures.libc.sscanf, 1360
angr.procedures.libc.stpcpy, 1360
angr.procedures.libc.strcat, 1360
angr.procedures.libc.strchr, 1360
angr.procedures.libc.strcmp, 1360
angr.procedures.libc.strcoll, 1360
angr.procedures.libc.strcpy, 1360
angr.procedures.libc.strlen, 1360
angr.procedures.libc.strncat, 1360
angr.procedures.libc.strncmp, 1360
angr.procedures.libc.strncpy, 1361
angr.procedures.libc.strnlen, 1361
angr.procedures.libc.strstr, 1361
angr.procedures.libc.strtok, 1361
angr.procedures.libc.strtol, 1361
angr.procedures.libc.strtoul, 1361
angr.procedures.libc.strxfrm, 1362
angr.procedures.libc.system, 1362
angr.procedures.libc.time, 1362
angr.procedures.libc.tmpnam, 1362
angr.procedures.libc.tolower, 1362
angr.procedures.libc.toupper, 1362
angr.procedures.libc.ungetc, 1362
angr.procedures.libc.vsnprintf, 1362
angr.procedures.libc.wchar, 1362
angr.procedures.libstdcpp, 1362
angr.procedures.libstdcpp.std____throw_bad_alloc, 1363
angr.procedures.libstdcpp.std____throw_bad_cast, 1363
angr.procedures.libstdcpp.std____throw_length_error, 1363
angr.procedures.libstdcpp.std____throw_logic_error, 1363
angr.procedures.libstdcpp.std__terminate, 1363
angr.procedures.linux_kernel, 1363
angr.procedures.linux_kernel.access, 1364
angr.procedures.linux_kernel.arch_prctl, 1364
angr.procedures.linux_kernel.arm_user_helpers, 1364
angr.procedures.linux_kernel.brk, 1364
angr.procedures.linux_kernel.cwd, 1365
angr.procedures.linux_kernel.fstat, 1365
angr.procedures.linux_kernel.fstat64, 1365
angr.procedures.linux_kernel.futex, 1365
angr.procedures.linux_kernel.getegid, 1365
angr.procedures.linux_kernel.geteuid, 1365
angr.procedures.linux_kernel.getgid, 1365
angr.procedures.linux_kernel.getpid, 1365
angr.procedures.linux_kernel.getrlimit, 1366
angr.procedures.linux_kernel.gettid, 1366
angr.procedures.linux_kernel.getuid, 1366
angr.procedures.linux_kernel.iovec, 1366
angr.procedures.linux_kernel.lseek, 1366
angr.procedures.linux_kernel.mmap, 1366
angr.procedures.linux_kernel.mprotect, 1366
angr.procedures.linux_kernel.munmap, 1366
angr.procedures.linux_kernel.openat, 1367
angr.procedures.linux_kernel.set_tid_address, 1367
angr.procedures.linux_kernel.sigaction, 1367
angr.procedures.linux_kernel.sigprocmask, 1367
angr.procedures.linux_kernel.stat, 1367
angr.procedures.linux_kernel.sysinfo, 1367
angr.procedures.linux_kernel.tgkill, 1367
angr.procedures.linux_kernel.time, 1367
angr.procedures.linux_kernel.uid, 1367
angr.procedures.linux_kernel.uname, 1368
angr.procedures.linux_kernel.unlink, 1368
angr.procedures.linux_kernel.vsyscall, 1368

angr.procedures.linux_loader, 1368
angr.procedures.linux_loader.sim_loader, 1368
angr.procedures.linux_loader.tls, 1368
angr.procedures.msvcr, 1369
angr.procedures.msvcr.fmode, 1369
angr.procedures.ntdll, 1369
angr.procedures.ntdll.exceptions, 1369
angr.procedures.posix, 1369
angr.procedures.posix.accept, 1370
angr.procedures.posix.bind, 1370
angr.procedures.posix.bzero, 1370
angr.procedures.posix.chroot, 1371
angr.procedures.posix.close, 1371
angr.procedures.posix.closedir, 1371
angr.procedures.posix.dup, 1371
angr.procedures.posix.fcntl, 1371
angr.procedures.posix.fdopen, 1371
angr.procedures.posix.fileno, 1371
angr.procedures.posix.fork, 1372
angr.procedures.posix.getenv, 1372
angr.procedures.posix.gethostbyname, 1372
angr.procedures.posix.getpass, 1372
angr.procedures.posix.getsockopt, 1372
angr.procedures.posix.htonl, 1372
angr.procedures.posix.htons, 1372
angr.procedures.posix.inet_ntoa, 1372
angr.procedures.posix.listen, 1373
angr.procedures.posix.mmap, 1373
angr.procedures.posix.open, 1373
angr.procedures.posix.opendir, 1373
angr.procedures.posix.poll, 1373
angr.procedures.posix.pread64, 1373
angr.procedures.posix.pthread, 1373
angr.procedures.posix.pwrite64, 1374
angr.procedures.posix.read, 1374
angr.procedures.posix.readdir, 1374
angr.procedures.posix.recv, 1374
angr.procedures.posix.recvfrom, 1374
angr.procedures.posix.select, 1374
angr.procedures.posix.send, 1374
angr.procedures.posix.setsockopt, 1375
angr.procedures.posix.sigaction, 1375
angr.procedures.posix.sim_time, 1375
angr.procedures.posix.sleep, 1375
angr.procedures.posix.socket, 1375
angr.procedures.posix.strcasecmp, 1375
angr.procedures.posix.strdup, 1375
angr.procedures.posix.strncasecmp, 1375
angr.procedures.posix.strndup, 1375
angr.procedures.posix.strtok_r, 1375
angr.procedures.posix.syslog, 1376
angr.procedures.posix.tz, 1376
angr.procedures.posix.unlink, 1376
angr.procedures.posix.usleep, 1376
angr.procedures.posix.write, 1376
angr.procedures.procedure_dict, 1376
angr.procedures.stubs, 1376
angr.procedures.stubs.b64_decode, 1378
angr.procedures.stubs.caller, 1378
angr.procedures.stubs.CallReturn, 1376
angr.procedures.stubs.crazy_scanf, 1378
angr.procedures.stubs.format_parser, 1378
angr.procedures.stubs.Nop, 1377
angr.procedures.stubs.NoReturnUnconstrained, 1377
angr.procedures.stubs.PathTerminator, 1377
angr.procedures.stubs.Redirect, 1377
angr.procedures.stubs.ReturnChar, 1377
angr.procedures.stubs.ReturnUnconstrained, 1377
angr.procedures.stubs.syscall_stub, 1380
angr.procedures.stubs.UnresolvableCallTarget, 1377
angr.procedures.stubs.UnresolvableJumpTarget, 1377
angr.procedures.stubs.UserHook, 1378
angr.procedures.testing, 1380
angr.procedures.testing.manyargs, 1380
angr.procedures.testing.retreg, 1380
angr.procedures.tracer, 1380
angr.procedures.tracer.random, 1381
angr.procedures.tracer.receive, 1381
angr.procedures.tracer.transmit, 1381
angr.procedures.uclibc, 1381
angr.procedures.win32, 1381
angr.procedures.win32.critical_section, 1383
angr.procedures.win32.dynamic_loading, 1383
angr.procedures.win32.EncodePointer, 1382
angr.procedures.win32.ExitProcess, 1382
angr.procedures.win32.file_handles, 1384
angr.procedures.win32.GetCommandLine, 1382
angr.procedures.win32.GetCurrentProcessId, 1382
angr.procedures.win32.GetCurrentThreadId, 1382
angr.procedures.win32.gethostbyname, 1384
angr.procedures.win32.GetLastInputInfo, 1382
angr.procedures.win32.GetModuleHandle, 1382
angr.procedures.win32.GetProcessAffinityMask, 1383
angr.procedures.win32.heap, 1384
angr.procedures.win32.InterlockedExchange, 1383
angr.procedures.win32.is_bad_ptr, 1384
angr.procedures.win32.IsProcessorFeaturePresent, 1383
angr.procedures.win32.local_storage, 1385
angr.procedures.win32.mutex, 1385

angr.procedures.win32.sim_time, 1386
angr.procedures.win32.system_paths, 1387
angr.procedures.win32.VirtualAlloc, 1383
angr.procedures.win32.VirtualProtect, 1383
angr.procedures.win32_kernel, 1387
angr.procedures.win32_kernel.ExAllocatePool, 1387
angr.procedures.win32_kernel.ExFreePoolWithTag, 1387
angr.procedures.win_user32, 1387
angr.procedures.win_user32.chars, 1387
angr.procedures.win_user32.keyboard, 1387
angr.procedures.win_user32.messagebox, 1388
angr.project, 1388
angr.protos, 1393
angr.protos.ail_types_pb2, 1393
angr.protos.cfg_pb2, 1395
angr.protos.clinic_pb2, 1396
angr.protos.codegen_pb2, 1397
angr.protos.decompilation_cache_pb2, 1402
angr.protos.function_pb2, 1402
angr.protos.primitives_pb2, 1403
angr.protos.variables_pb2, 1405
angr.protos.xrefs_pb2, 1406
angr.rust, 1406
angr.rust.analyses, 1406
angr.rust.analyses.cleanup_function_identification, 1406
angr.rust.analyses.flirt_sig_propagation, 1407
angr.rust.analyses.rust_calling_convention, 1407
angr.rust.analyses.rust_calling_convention.fact_collector, 1408
angr.rust.analyses.rust_calling_convention.pathfinder, 1409
angr.rust.analyses.rust_calling_convention.rust_calling_convention, 1409
angr.rust.analyses.rust_calling_convention.rust_calling_convention_model, 1410
angr.rust.analyses.rust_symbol_recovery, 1410
angr.rust.analyses.rust_type_hints, 1410
angr.rust.analyses.rustc_version_identification, 1410
angr.rust.analyses.struct_builder, 1410
angr.rust.analyses.type_db_loader, 1411
angr.rust.knowledge_plugins, 1411
angr.rust.knowledge_plugins.clinic_factory, 1412
angr.rust.knowledge_plugins.known_structs, 1412
angr.rust.knowledge_plugins.librust, 1413
angr.rust.knowledge_plugins.rust_calling_conventions, 1413
angr.rust.knowledge_plugins.type_hints, 1413
angr.rust.mixins, 1413
angr.rust.mixins.cfa_mixin, 1417
angr.rust.mixins.cfg_transformation_mixin, 1418
angr.rust.mixins.dfa_mixin, 1418
angr.rust.mixins.srda_mixin, 1419
angr.rust.mixins.ssa_variable_mixin, 1420
angr.rust.optimization_passes, 1421
angr.rust.optimization_passes.cleanup_code_remover, 1421
angr.rust.optimization_passes.combo_register_rewriter, 1421
angr.rust.optimization_passes.deref_coercion_simplifier, 1422
angr.rust.optimization_passes.deref_coercion_simplifier_ur 1422
angr.rust.optimization_passes.error_propagation_simplifier 1423
angr.rust.optimization_passes.function_prototype_inference 1423
angr.rust.optimization_passes.macro, 1423
angr.rust.optimization_passes.macro.format_macro_simplifie 1423
angr.rust.optimization_passes.outliners, 1424
angr.rust.optimization_passes.outliners.string_cmp_outline 1424
angr.rust.optimization_passes.outliners.string_literal_out 1425
angr.rust.optimization_passes.outliners.string_outliner, 1425
angr.rust.optimization_passes.outliners.unwrap_outliner, 1425
angr.rust.optimization_passes.outliners.vec_outliner, 1426
angr.rust.optimization_passes.pattern_match_simplifier, 1426
angr.rust.optimization_passes.pre_pattern_match_simplifier 1426
angr.rust.optimization_passes.redundant_block_remover, 1428
angr.rust.optimization_passes.ret_expr_rewriter, 1428
angr.rust.optimization_passes.rust_calling_convention, 1428
angr.rust.optimization_passes.security_check_remover, 1428
angr.rust.optimization_passes.str_argument_simplifier, 1429
angr.rust.optimization_passes.struct_instantiation_simplif 1429
angr.rust.optimization_passes.struct_return_simplifier, 1430
angr.rust.optimization_passes.utils, 1430

angr.rust.sim_type, 1431
angr.rust.structuring, 1435
angr.rust.structuring.structurer_nodes, 1436
angr.rust.typehoon, 1436
angr.rust.typehoon.translator, 1436
angr.rust.typehoon.typehoon, 1437
angr.rust.utils, 1437
angr.rust.utils.ail, 1437
angr.rust.utils.demangler, 1438
angr.rust.utils.rust_sigs, 1439
angr.rustylib, 1439
angr.serializable, 1440
angr.sim_manager, 1441
angr.sim_options, 1448
angr.sim_procedure, 1448
angr.sim_state, 1452
angr.sim_state_options, 1456
angr.sim_type, 1459
angr.sim_variable, 1477
angr.simos, 1479
angr.simos.cgc, 1486
angr.simos.javavm, 1486
angr.simos.linux, 1488
angr.simos.simos, 1488
angr.simos.snimmuc_nxp, 1491
angr.simos.uefi, 1491
angr.simos.userland, 1491
angr.simos.windows, 1492
angr.simos.xbox, 1493
angr.slicer, 1493
angr.state_hierarchy, 1494
angr.state_plugins, 1494
angr.state_plugins.callstack, 1542
angr.state_plugins.cgc, 1544
angr.state_plugins.debug_variables, 1545
angr.state_plugins.edge_hitmap, 1547
angr.state_plugins.filesystem, 1547
angr.state_plugins.gdb, 1551
angr.state_plugins.globals, 1551
angr.state_plugins.heap, 1552
angr.state_plugins.heap.heap_base, 1555
angr.state_plugins.heap.heap_brk, 1556
angr.state_plugins.heap.heap_freelist, 1557
angr.state_plugins.heap.heap_libc, 1558
angr.state_plugins.heap.heap_ptmalloc, 1559
angr.state_plugins.heap.utils, 1561
angr.state_plugins.history, 1561
angr.state_plugins.icicle, 1563
angr.state_plugins.inspect, 1565
angr.state_plugins.javavm_classloader, 1573
angr.state_plugins.jni_references, 1573
angr.state_plugins.libc, 1574
angr.state_plugins.light_registers, 1577
angr.state_plugins.log, 1577
angr.state_plugins.loop_data, 1577
angr.state_plugins.plugin, 1578
angr.state_plugins.posix, 1579
angr.state_plugins.preconstrainer, 1582
angr.state_plugins.scratch, 1583
angr.state_plugins.sim_action, 1584
angr.state_plugins.sim_action_object, 1585
angr.state_plugins.sim_event, 1587
angr.state_plugins.solver, 1587
angr.state_plugins.symbolizer, 1596
angr.state_plugins.trace_additions, 1596
angr.state_plugins.uc_manager, 1598
angr.state_plugins.unicorn_engine, 1599
angr.state_plugins.view, 1604
angr.storage, 1607
angr.storage.file, 1609
angr.storage.memory_mixins, 1617
angr.storage.memory_mixins.actions_mixin, 1633
angr.storage.memory_mixins.address_concretization_mixin, 1633
angr.storage.memory_mixins.bvv_conversion_mixin, 1634
angr.storage.memory_mixins.conditional_store_mixin, 1634
angr.storage.memory_mixins.convenient_mappings_mixin, 1634
angr.storage.memory_mixins.default_filler_mixin, 1635
angr.storage.memory_mixins.dirty_addrs_mixin, 1635
angr.storage.memory_mixins.hex_dumper_mixin, 1635
angr.storage.memory_mixins.inspect_mixin, 1636
angr.storage.memory_mixins.javavm_memory_mixin, 1636
angr.storage.memory_mixins.keyvalue_memory_mixin, 1637
angr.storage.memory_mixins.label_merger_mixin, 1638
angr.storage.memory_mixins.memory_mixin, 1638
angr.storage.memory_mixins.multi_value_merger_mixin, 1641
angr.storage.memory_mixins.name_resolution_mixin, 1641
angr.storage.memory_mixins.paged_memory, 1641
angr.storage.memory_mixins.paged_memory.page_backer_mixins 1641
angr.storage.memory_mixins.paged_memory.paged_memory_mixin 1641
angr.storage.memory_mixins.paged_memory.paged_memory_multi 1643
angr.storage.memory_mixins.paged_memory.pages,

1643
angr.storage.memory_mixins.paged_memory.pages.base, 1647
angr.storage.memory_mixins.paged_memory.pages.cooperation, 1647
angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin, 1648
angr.storage.memory_mixins.paged_memory.pages.ispo_mixin, 1648
angr.storage.memory_mixins.paged_memory.pages.list_page, 1649
angr.storage.memory_mixins.paged_memory.pages.multi_values, 1649
angr.storage.memory_mixins.paged_memory.pages.mvl_utils_ae, 1651
angr.storage.memory_mixins.paged_memory.pages.permissions_mixin, 1651
angr.storage.memory_mixins.paged_memory.pages.refcount_mixin, 1652
angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap, 1652
angr.storage.memory_mixins.paged_memory.pages.ultra_page, 1654
angr.storage.memory_mixins.paged_memory.privileged_mixin, 1655
angr.storage.memory_mixins.paged_memory.stack_allocation_mixin, 1655
angr.storage.memory_mixins.regioned_memory, 1656
angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor, 1659
angr.storage.memory_mixins.regioned_memory.abstract_merger_mixin, 1659
angr.storage.memory_mixins.regioned_memory.region_utils_library, 1659
angr.storage.memory_mixins.regioned_memory.region_data, 1659
angr.storage.memory_mixins.regioned_memory.region_meta_mixin, 1661
angr.storage.memory_mixins.regioned_memory.region_utils_ssa_combined_uses_collector, 1662
angr.storage.memory_mixins.regioned_memory.region_utils_ssa_vvar, 1663
angr.storage.memory_mixins.regioned_memory.static_find_mixin, 1664
angr.storage.memory_mixins.simple_interface_mixin, 1664
angr.storage.memory_mixins.simplification_mixin, 1665
angr.storage.memory_mixins.size_resolution_mixin, 1665
angr.storage.memory_mixins.slotted_memory, 1665
angr.storage.memory_mixins.smart_find_mixin, 1666
angr.storage.memory_mixins.symbolic_merger_mixin, 1666
angr.storage.memory_mixins.top_merger_mixin, 1666
angr.storage.memory_mixins.underconstrained_mixin, 1666
angr.storage.memory_mixins.unwrapper_mixin, 1666
angr.storage.memory_object, 1666
angr.tablespecs, 1668
angr.utils, 1668
angr.utils.ail, 1669
angr.utils.ail_serialization, 1671
angr.utils.algo, 1673
angr.utils.balancer, 1673
angr.utils.bits, 1674
angr.utils.constants, 1675
angr.utils.cowdict, 1676
angr.utils.cmp, 1677
angr.utils.doms, 1677
angr.utils.dynamic_dictlist, 1678
angr.utils.endness, 1678
angr.utils.enums_conv, 1679
angr.utils.env, 1679
angr.utils.formatting, 1679
angr.utils.funcid, 1680
angr.utils.go_runtime, 1680
angr.utils.graph, 1682
angr.utils.ins_addr_descriptor, 1689
angr.utils.ins_addr_list, 1689
angr.utils.jsonxutils, 1690
angr.utils.lazy_import, 1691
angr.utils.library, 1691
angr.utils.loader, 1693
angr.utils.mp, 1693
angr.utils.orderedset, 1695
angr.utils.smart_cache, 1695
angr.utils.ssa, 1695
angr.utils.ssa.combined_uses_collector, 1698
angr.utils.ssa.tmp_uses_collector, 1699
angr.utils.ssa.vvar_extra_defs_collector, 1699
angr.utils.ssa.vvar_uses_collector, 1699
angr.utils.strings, 1699
angr.utils.tagged_interval_map, 1700
angr.utils.timing, 1700
angr.utils.types, 1701
angr.utils.vex, 1703
angr.vaults, 1704

# INDEX

## Symbols


__init__() (*angr.ailment.block_walker.AILBlockRewriter method*), 216
__init__() (*angr.ailment.block_walker.AILBlockWalker method*), 215
__init__() (*angr.ailment.converter_pcode.PCodeIRSBConverter method*), 217
__init__() (*angr.ailment.expression.Expression method*), 220
__init__() (*angr.ailment.statement.Statement method*), 226
__init__() (*angr.analyses.analysis.AnalysesHub method*), 275
__init__() (*angr.analyses.analysis.AnalysisFactory method*), 276
__init__() (*angr.analyses.analysis.AnalysisLogEntry method*), 275
__init__() (*angr.analyses.backward_slice.BackwardSlice method*), 277
__init__() (*angr.analyses.binary_optimizer.BinaryOptimizer method*), 279
__init__() (*angr.analyses.binary_optimizer.ConstantPropagation method*), 279
__init__() (*angr.analyses.binary_optimizer.DeadAssignment method*), 279
__init__() (*angr.analyses.binary_optimizer.RedundantStackVariable method*), 279
__init__() (*angr.analyses.binary_optimizer.RegisterReallocation method*), 279
__init__() (*angr.analyses.bindiff.BinDiff method*), 282
__init__() (*angr.analyses.bindiff.ConstantChange method*), 280
__init__() (*angr.analyses.bindiff.Difference method*), 280
__init__() (*angr.analyses.bindiff.FunctionDiff method*), 280
__init__() (*angr.analyses.bindiff.NormalizedBlock method*), 280
__init__() (*angr.analyses.bindiff.NormalizedFunction method*), 280
__init__() (*angr.analyses.boyscout.BoyScout method*), 283
__init__() (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder method*), 283
__init__() (*angr.analyses.calling_convention.calling_convention.CallSite method*), 285
__init__() (*angr.analyses.calling_convention.calling_convention.Calling method*), 286
__init__() (*angr.analyses.calling_convention.fact_collector.FactCollecto method*), 288
__init__() (*angr.analyses.calling_convention.fact_collector.FactCollecto method*), 287
__init__() (*angr.analyses.calling_convention.fact_collector.SimEngineFa method*), 288
__init__() (*angr.analyses.cdg.CDG method*), 289
__init__() (*angr.analyses.cfg.cfb.CFBlanket method*), 305
__init__() (*angr.analyses.cfg.cfb.CFBlanketView method*), 304
__init__() (*angr.analyses.cfg.cfb.MemoryRegion method*), 304
__init__() (*angr.analyses.cfg.cfb.Unknown method*), 304
__init__() (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions method*), 307
__init__() (*angr.analyses.cfg.cfg_base.CFGBase method*), 307
__init__() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 311
__init__() (*angr.analyses.cfg.cfg_emulated.CFGJob method*), 310
__init__() (*angr.analyses.cfg.cfg_emulated.PendingJob method*), 311
__init__() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 321
__init__() (*angr.analyses.cfg.cfg_fast.CFGJob method*), 320
__init__() (*angr.analyses.cfg.cfg_fast.CFGResumeState method*), 318
__init__() (*angr.analyses.cfg.cfg_fast.DecodingAssumption method*), 315
__init__() (*angr.analyses.cfg.cfg_fast.FunctionCallEdge method*), 319
__init__() (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge

method), 319
__init__() (angr.analyses.cfg.cfg_fast.FunctionReturn method), 315
__init__() (angr.analyses.cfg.cfg_fast.FunctionReturnEdge method), 319
__init__() (angr.analyses.cfg.cfg_fast.FunctionTransitionEdge method), 318
__init__() (angr.analyses.cfg.cfg_fast.PendingJobs method), 316
__init__() (angr.analyses.cfg.cfg_job_base.CFGJobBase method), 326
__init__() (angr.analyses.cfg.cfg_job_base.FunctionKey method), 326
__init__() (angr.analyses.cfg.indirect_jump_resolvers.aarch64_elf_got.AArch64ElfGotResolver method), 331
__init__() (angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got.AMD64ElfGotResolver method), 331
__init__() (angr.analyses.cfg.indirect_jump_resolvers.amd64_jumptable.AMD64JumpTableResolver method), 332
__init__() (angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.ArmElfFastResolver method), 332
__init__() (angr.analyses.cfg.indirect_jump_resolvers.constant_resolver.ConstantResolver method), 332
__init__() (angr.analyses.cfg.indirect_jump_resolvers.constant_value_manager.ConstantValueManager method), 333
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransferringAnnotation method), 334
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method), 337
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessor method), 336
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableProcessorState method), 335
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTableResolver method), 337
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTargetBaseAddr method), 335
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.LocalHook method), 336
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHook method), 337
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation method), 335
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegisterInitializedHook method), 336
__init__() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.Tmp method), 334
__init__() (angr.analyses.cfg.indirect_jump_resolvers.mem_load_resolver.MemoryLoadResolver method), 338
__init__() (angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.MipsElfFastResolver method), 339
__init__() (angr.analyses.cfg.indirect_jump_resolvers.mips_elf_got.MipsElfGotResolver method), 339
__init__() (angr.analyses.cfg.indirect_jump_resolvers.propagator_utils.PropagatorLoadCallback method), 339
__init__() (angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJu method), 340
__init__() (angr.analyses.cfg.indirect_jump_resolvers.syscall_resolver.Sy method), 340
__init__() (angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt.X8 method), 341
__init__() (angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat.X86Pe method), 341
__init__() (angr.analyses.cfg.pe_msvc_eh_structs.EH4ScopeRecord method), 344
__init__() (angr.analyses.cfg.pe_msvc_eh_structs.EH4ScopeTable method), 343
__init__() (angr.analyses.cfg.pe_msvc_eh_structs.FuncInfo method), 342
__init__() (angr.analyses.cfg.pe_msvc_eh_structs.HandlerType method), 345
__init__() (angr.analyses.cfg.pe_msvc_eh_structs.TryBlockMapEntry method), 345
__init__() (angr.analyses.cfg.pe_msvc_eh_structs.UnwindMapEntry method), 343
__init__() (angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceT method), 348
__init__() (angr.analyses.class_identifier.ClassIdentifier method), 351
__init__() (angr.analyses.code_tagging.CodeTagging method), 351
__init__() (angr.analyses.codecave.CodeCave method), 352
__init__() (angr.analyses.codecave.CodeCaveAnalysis method), 352
__init__() (angr.analyses.complete_calling_conventions.CompleteCallin method), 353
__init__() (angr.analyses.congruency_check.CongruencyCheck method), 354
__init__() (angr.analyses.data_dep.data_dependency_analysis.DataDep method), 358
__init__() (angr.analyses.data_dep.data_dependency_analysis.NodalAn method), 358
__init__() (angr.analyses.data_dep.dep_nodes.BaseDepNode method), 359
__init__() (angr.analyses.data_dep.dep_nodes.ConstantDepNode method), 359
__init__() (angr.analyses.data_dep.dep_nodes.MemDepNode method), 359
__init__() (angr.analyses.data_dep.dep_nodes.RegDepNode method), 360
__init__() (angr.analyses.data_dep.dep_nodes.TmpDepNode method), 360
__init__() (angr.analyses.data_dep.dep_nodes.VarDepNode method), 360
__init__() (angr.analyses.data_dep.sim_act_location.ParsedInstruction method), 361
__init__() (angr.analyses.data_dep.sim_act_location.SimActLocation

method), 361
__init__() (angr.analyses.datagraph_meta.DataGraphMeta method), 361
__init__() (angr.analyses.ddg.AST method), 361
__init__() (angr.analyses.ddg.DDG method), 364
__init__() (angr.analyses.ddg.DDGJob method), 362
__init__() (angr.analyses.ddg.DDGView method), 364
__init__() (angr.analyses.ddg.DDGViewInstruction method), 364
__init__() (angr.analyses.ddg.DDGViewItem method), 364
__init__() (angr.analyses.ddg.LiveDefinitions method), 362
__init__() (angr.analyses.ddg.ProgramVariable method), 362
__init__() (angr.analyses.decompiler.ail_simplifier.AILBlockTempCollector method), 397
__init__() (angr.analyses.decompiler.ail_simplifier.AILSimplifier method), 397
__init__() (angr.analyses.decompiler.ail_simplifier.InsertLoadStoreRegister method), 397
__init__() (angr.analyses.decompiler.ail_simplifier.PartialConstantPropagator method), 397
__init__() (angr.analyses.decompiler.ailgraph_walker.AILGraphWalker method), 398
__init__() (angr.analyses.decompiler.block_io_finder.BlockIOFinder method), 398
__init__() (angr.analyses.decompiler.block_simplifier.BlockSimplifier method), 400
__init__() (angr.analyses.decompiler.block_simplifier.PeepholeOptimizationBundle method), 399
__init__() (angr.analyses.decompiler.callsite_maker.CallSiteMaker method), 401
__init__() (angr.analyses.decompiler.ccall_rewriters.rewriter_base.CCallRewriterBase method), 402
__init__() (angr.analyses.decompiler.clinic.Clinic method), 405
__init__() (angr.analyses.decompiler.clinic.ComboRegReferenceWalker method), 405
__init__() (angr.analyses.decompiler.clinic.DataRefDesc method), 404
__init__() (angr.analyses.decompiler.condition_processor.ConditionProcessor method), 408
__init__() (angr.analyses.decompiler.counters.boolean_counter.BooleanCounter method), 411
__init__() (angr.analyses.decompiler.counters.call_counter.AILBlockCallCounter method), 411
__init__() (angr.analyses.decompiler.counters.call_counter.AILCallCounter method), 411
__init__() (angr.analyses.decompiler.counters.expression_counters.OperationCounter method), 412
__init__() (angr.analyses.decompiler.counters.expression_counters.RegisterExpressionCounter method), 412
__init__() (angr.analyses.decompiler.counters.expression_counters.SingleExpressionCounter method), 412
__init__() (angr.analyses.decompiler.counters.seq_cf_structure_counter.SequenceCFStructureCounter method), 412
__init__() (angr.analyses.decompiler.decompilation_cache.DecompilationCache method), 412
__init__() (angr.analyses.decompiler.decompilation_options.DecompilationOption method), 413
__init__() (angr.analyses.decompiler.decompiler.ConstantCollector method), 414
__init__() (angr.analyses.decompiler.decompiler.Decompiler method), 415
__init__() (angr.analyses.decompiler.dephication.dephication_base.DephicationBase method), 418
__init__() (angr.analyses.decompiler.dephication.graph_dephication.GraphDephication method), 419
__init__() (angr.analyses.decompiler.dephication.graph_vvar_mapping.GraphVVarMapping method), 419
__init__() (angr.analyses.decompiler.dephication.rewriting_engine.SimEngineDephicationVEX method), 420
__init__() (angr.analyses.decompiler.dephication.seqnode_dephication.DephicationRewriter method), 420
__init__() (angr.analyses.decompiler.dephication.seqnode_dephication.SeqNodeDephication method), 420
__init__() (angr.analyses.decompiler.dephication.seqnode_dephication.SeqNodeDephicationVEX method), 420
__init__() (angr.analyses.decompiler.dirty_rewriters.rewriter_base.DirtyRewriterBase method), 421
__init__() (angr.analyses.decompiler.edits.resolve.ResolvedVariable method), 448
__init__() (angr.analyses.decompiler.edits.results.EditResult method), 450
__init__() (angr.analyses.decompiler.edits.results.Refresh method), 449
__init__() (angr.analyses.decompiler.empty_node_remover.EmptyNodeRemover method), 450
__init__() (angr.analyses.decompiler.expression_narrower.EffectiveSizeExpressionNarrower method), 451
__init__() (angr.analyses.decompiler.expression_narrower.ExprNarrowingWalker method), 451
__init__() (angr.analyses.decompiler.expression_narrower.ExpressionNarrower method), 451
__init__() (angr.analyses.decompiler.goto_manager.Goto method), 452
__init__() (angr.analyses.decompiler.goto_manager.GotoManager method), 452
__init__() (angr.analyses.decompiler.jump_target_collector.JumpTargetCollector method), 453
__init__() (angr.analyses.decompiler.jumptable_entry_condition_rewriter.JumpTableEntryConditionRewriter method), 453
__init__() (angr.analyses.decompiler.known_patterns.context.PatternContext method), 489
__init__() (angr.analyses.decompiler.known_patterns.dsl.MatchCtx method), 491
__init__() (angr.analyses.decompiler.known_patterns.dsl.MatchState

*method*), 490
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PAny method*), 492
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PAssign method*), 503
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PBinOp method*), 494
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PBlankPat method*), 505
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PCall method*), 497
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PCallResult method*), 498
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PCallStmt method*), 504
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PChoice method*), 502
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PConvert method*), 504
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PConst method*), 493
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PCont method*), 495
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PDefOf method*), 498
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PExtract method*), 496
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PField method*), 499
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PGraftPoint method*), 506
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PIfThen method*), 501
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PLoad method*), 496
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PPhi method*), 501
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PStackField method*), 500
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PStmtSeq method*), 505
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PStore method*), 503
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PUnaryOp method*), 495
__init__() (*angr.analyses.decompiler.known_patterns.dsl.PVar method*), 493
__init__() (*angr.analyses.decompiler.known_patterns.finder.KnownPatternFinder method*), 509
__init__() (*angr.analyses.decompiler.known_patterns.finder.KnownPatternMatch method*), 507
__init__() (*angr.analyses.decompiler.known_patterns.finder.OutlineResult method*), 508
__init__() (*angr.analyses.decompiler.known_patterns.gating.AllOf method*), 514
__init__() (*angr.analyses.decompiler.known_patterns.gating.AnyOf method*), 513
__init__() (*angr.analyses.decompiler.known_patterns.gating.CorroborateAll method*), 513
__init__() (*angr.analyses.decompiler.known_patterns.gating.CorroborateAny method*), 514
__init__() (*angr.analyses.decompiler.known_patterns.gating.GateContext method*), 511
__init__() (*angr.analyses.decompiler.known_patterns.gating.PatternGate method*), 512
__init__() (*angr.analyses.decompiler.known_patterns.gating.TargetGate method*), 512
__init__() (*angr.analyses.decompiler.known_patterns.generator.PatternCollector method*), 515
__init__() (*angr.analyses.decompiler.known_patterns.pattern.CppRef method*), 519
__init__() (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern method*), 522
__init__() (*angr.analyses.decompiler.known_patterns.pattern.PatternParam method*), 520
__init__() (*angr.analyses.decompiler.known_patterns.templates.KnownPattern method*), 532
__init__() (*angr.analyses.decompiler.known_patterns.wdk_shared_data.WdkSharedData method*), 534
__init__() (*angr.analyses.decompiler.label_collector.LabelCollector method*), 535
__init__() (*angr.analyses.decompiler.node_replacer.NodeReplacer method*), 535
__init__() (*angr.analyses.decompiler.notes.decompilation_note.DecompilationNote method*), 538
__init__() (*angr.analyses.decompiler.notes.deobfuscated_strings.DeobfuscatedStrings method*), 539
__init__() (*angr.analyses.decompiler.notes.deobfuscated_strings.DeobfuscatedStringsAnalysis method*), 539
__init__() (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePointerSaveSimplifier method*), 560
__init__() (*angr.analyses.decompiler.optimization_passes.call_stmt_rewriter.CallStatementRewriter method*), 560
__init__() (*angr.analyses.decompiler.optimization_passes.code_motion.CodeMotionOptimizationPass method*), 561
__init__() (*angr.analyses.decompiler.optimization_passes.condition_constprop.ConditionConstantPropagation method*), 562
__init__() (*angr.analyses.decompiler.optimization_passes.condition_constprop.ConditionConstantPropagationOptimizationPass method*), 562
__init__() (*angr.analyses.decompiler.optimization_passes.condition_constprop.ConditionConstantPropagationRewriter method*), 562
__init__() (*angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDereferenceAnalysis method*), 562
__init__() (*angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDereferencesSimplification method*), 563
__init__() (*angr.analyses.decompiler.optimization_passes.const_prop_reverter.ConstantPropagationRevertingOptimizationPass method*), 563
__init__() (*angr.analyses.decompiler.optimization_passes.const_prop_reverter.ConstantPropagationReverter

*method*), 564
__init__() (*angr.analyses.decompiler.optimization_passes.cmp_zero_negation_transformer.CmpZeroNegationTransformer method*), 564
__init__() (*angr.analyses.decompiler.optimization_passes.deadblock_remover.DeadblockRemover method*), 565
__init__() (*angr.analyses.decompiler.optimization_passes.determine_loadarg_indirect_vvars.DetermineLoadArgIndirectVVars method*), 565
__init__() (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier method*), 566
__init__() (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifierAddEngine method*), 565
__init__() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.DuplicationReverter method*), 568
__init__() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.EdgeGroupAnalysis method*), 569
__init__() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.EdgeGroupMerger method*), 570
__init__() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILEngine method*), 574
__init__() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILState method*), 574
__init__() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifyAILCallback method*), 574
__init__() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifyAILCallbackResult method*), 574
__init__() (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.BasePtrGVOpSwapper method*), 576
__init__() (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.OpDescriptor method*), 575
__init__() (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.StackCanonicalizer method*), 575
__init__() (*angr.analyses.decompiler.optimization_passes.flag_setting_cond_patterns.FlagSettingCondPatterns method*), 577
__init__() (*angr.analyses.decompiler.optimization_passes.flag_setting_cond_patterns.FlagSettingCondPatternsAdder method*), 576
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_memcpy_simplifier.InlinedMemcpySimplifier method*), 577
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_memset_simplifier.InlinedMemsetSimplifier method*), 577
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_memset_simplifier.InlinedMemsetSimplifierAddEngine method*), 578
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier.InlinedStrcpySimplifier method*), 579
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier method*), 581
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifierAddEngine method*), 580
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifierState method*), 581
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_strlen_simplifier.InlinedStrlenSimplifier method*), 579
__init__() (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier.InlinedWcscpySimplifier method*), 581
__init__() (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter method*), 582
__init__() (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverterAddEngine method*), 582
__init__() (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverterState method*), 582
__init__() (*angr.analyses.decompiler.optimization_passes.ite_region_converter.ITERegionConverter method*), 583
__init__() (*angr.analyses.decompiler.optimization_passes.known_pattern_handlers.KnownPatternHandlers method*), 583
__init__() (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier method*), 584
__init__() (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifierAddEngine method*), 584
__init__() (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifierState method*), 585
__init__() (*angr.analyses.decompiler.optimization_passes.mips_gp_setting.MipsGpSetting method*), 584
__init__() (*angr.analyses.decompiler.optimization_passes.mips_gp_setting.MipsGpSettingAddEngine method*), 585
__init__() (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier method*), 586
__init__() (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass method*), 587
__init__() (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage method*), 587
__init__() (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage.value method*), 588
__init__() (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage.value.__new__ method*), 589
__init__() (*angr.analyses.decompiler.optimization_passes.peephole_simplifier_multi_stage.PeepholeSimplifierMultiStage method*), 590
__init__() (*angr.analyses.decompiler.optimization_passes.register_save_simplifier.RegisterSaveSimplifier method*), 590
__init__() (*angr.analyses.decompiler.optimization_passes.register_save_simplifier.RegisterSaveSimplifierAddEngine method*), 591
__init__() (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier method*), 591
__init__() (*angr.analyses.decompiler.optimization_passes.ret_deduplicator.RetDeduplicator method*), 592
__init__() (*angr.analyses.decompiler.optimization_passes.return_duplicator_base.ReturnDuplicatorBase method*), 592
__init__() (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.CodeReplicator method*), 592
__init__() (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.CodeReplicatorAILEngine method*), 593
__init__() (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.CodeReplicatorAILState method*), 594
__init__() (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.CodeReplicatorCallback method*), 594
__init__() (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.CodeReplicatorCallbackResult

method), 594
__init__() (angr.analyses.decompiler.optimization_passes.simplify_function_call_replacements.SimplifyFunctionCallReplacements method), 595
__init__() (angr.analyses.decompiler.optimization_passes.simplify_function_call_replacements.SimplifyFunctionCallReplacementsBase method), 595
__init__() (angr.analyses.decompiler.optimization_passes.simplify_function_call_replacements.SimplifyFunctionCallReplacementsRegistry method), 596
__init__() (angr.analyses.decompiler.optimization_passes.simplify_function_call_replacements.SimplifyFunctionCallReplacementsRegistryMeta method), 595
__init__() (angr.analyses.decompiler.optimization_passes.simplify_function_call_replacements.SimplifyFunctionCallReplacementsRegistryNamespace method), 595
__init__() (angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator.SwitchDefaultCaseDuplicator method), 596
__init__() (angr.analyses.decompiler.optimization_passes.switch_reused_entry_rewriter.SwitchReusedEntryRewriter method), 597
__init__() (angr.analyses.decompiler.optimization_passes.tag_slicer.TagSlicer method), 598
__init__() (angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier.WinStackCanarySimplifier method), 598
__init__() (angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier method), 598
__init__() (angr.analyses.decompiler.peephole_optimizations.PeepholeOptimizationExprBase method), 604
__init__() (angr.analyses.decompiler.peephole_optimizations.PeepholeOptimizationMultiStmtBase method), 603
__init__() (angr.analyses.decompiler.peephole_optimizations.PeepholeOptimizationStmtBase method), 602
__init__() (angr.analyses.decompiler.presets.preset.DecompilationPreset method), 622
__init__() (angr.analyses.decompiler.redundant_label_remover.RedundantLabelRemover method), 623
__init__() (angr.analyses.decompiler.region_identifier.RegionIdentifier method), 623
__init__() (angr.analyses.decompiler.region_overlay.OverlayManager method), 624
__init__() (angr.analyses.decompiler.region_overlay.RegionOverlay method), 625
__init__() (angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer.CascadingConditionTransformer method), 633
__init__() (angr.analyses.decompiler.region_simplifiers.cascading_ifs.CascadingIfsRemover method), 634
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ConstantDoubleExpr method), 635
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ConstantFloatExpr method), 635
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionCounter method), 636
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionFolder method), 637
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionReplacer method), 634
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.ExpressionSpotter method), 637
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.FixedBufferPropagator method), 636
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.InlinedStringTransformer method), 636
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.LocalVariableReplacer method), 636
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.StackVariableSubstitutor method), 635
__init__() (angr.analyses.decompiler.region_simplifiers.expr_folding.StoreStackVariableSimplifier method), 635
__init__() (angr.analyses.decompiler.region_simplifiers.goto.GotoSimplifier method), 634
__init__() (angr.analyses.decompiler.region_simplifiers.if_.IfSimplifier method), 637
__init__() (angr.analyses.decompiler.region_simplifiers.ifelse.IfElseFlattener method), 637
__init__() (angr.analyses.decompiler.region_simplifiers.loop.LoopSimplifier method), 638
__init__() (angr.analyses.decompiler.region_simplifiers.node_address_finder.NodeAddressFinder method), 638
__init__() (angr.analyses.decompiler.region_simplifiers.region_simplifier.RegionSimplifier method), 638
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifier method), 639
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifierBase method), 641
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifierRegistry method), 639
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifierRegistryMeta method), 639
__init__() (angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.SwitchClusterSimplifierRegistryNamespace method), 640
__init__() (angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier.SwitchExpressionSimplifier method), 642
__init__() (angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier.SwitchExpressionSimplifierWalker method), 642
__init__() (angr.analyses.decompiler.return_maker.ReturnMaker method), 642
__init__() (angr.analyses.decompiler.semantic_naming.array_index_naming.ArrayIndexNaming method), 647
__init__() (angr.analyses.decompiler.semantic_naming.boolean_naming.BooleanNaming method), 647
__init__() (angr.analyses.decompiler.semantic_naming.call_result_naming.CallResultNaming method), 648
__init__() (angr.analyses.decompiler.semantic_naming.naming_base.ClinitNaming method), 649
__init__() (angr.analyses.decompiler.semantic_naming.naming_base.RegisterNaming method), 649
__init__() (angr.analyses.decompiler.semantic_naming.naming_base.SemanticNamingBase method), 648
__init__() (angr.analyses.decompiler.semantic_naming.orchestrator.SemanticNamingOrchestrator

*method*), 650
__init__() (*angr.analyses.decompiler.semantic_naming.pointer_tracker.PointerTracker method*), 651
__init__() (*angr.analyses.decompiler.semantic_naming.region_identifier.RegionIdentifier method*), 651
__init__() (*angr.analyses.decompiler.semantic_naming.sizing.Sizing method*), 652
__init__() (*angr.analyses.decompiler.seq_to_blocks.SequenceToBlocks method*), 652
__init__() (*angr.analyses.decompiler.sequence_walker.SequenceWalker method*), 652
__init__() (*angr.analyses.decompiler.ssailification.rewriting.RewritingAnalysis method*), 653
__init__() (*angr.analyses.decompiler.ssailification.rewriting_engine.SimEngineSSARewriting method*), 654
__init__() (*angr.analyses.decompiler.ssailification.rewriting_state.RewritingState method*), 654
__init__() (*angr.analyses.decompiler.ssailification.ssailification.Ssailification method*), 655
__init__() (*angr.analyses.decompiler.ssailification.traversal.TraversalAnalysis method*), 655
__init__() (*angr.analyses.decompiler.ssailification.traversal_engine_vex.SimEngineSSATraversal method*), 656
__init__() (*angr.analyses.decompiler.ssailification.traversal_state.TraversalState method*), 656
__init__() (*angr.analyses.decompiler.stack_item.StackItem method*), 658
__init__() (*angr.analyses.decompiler.stackarg_offset_manager.StackArgOffsetManager method*), 659
__init__() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 660
__init__() (*angr.analyses.decompiler.structured_codegen.base.InstructionMapping method*), 669
__init__() (*angr.analyses.decompiler.structured_codegen.base.InstructionMappingElement method*), 669
__init__() (*angr.analyses.decompiler.structured_codegen.base.PositionMapping method*), 669
__init__() (*angr.analyses.decompiler.structured_codegen.base.PositionMappingElement method*), 668
__init__() (*angr.analyses.decompiler.structured_codegen.c.CAILBlock method*), 668
__init__() (*angr.analyses.decompiler.structured_codegen.c.CArrayType method*), 674
__init__() (*angr.analyses.decompiler.structured_codegen.c.CAssignment method*), 684
__init__() (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp method*), 676
__init__() (*angr.analyses.decompiler.structured_codegen.c.CBreak method*), 680
__init__() (*angr.analyses.decompiler.structured_codegen.c.CClosingObject method*), 675
__init__() (*angr.analyses.decompiler.structured_codegen.c.CConstant method*), 684
__init__() (*angr.analyses.decompiler.structured_codegen.c.CConstruct method*), 681
__init__() (*angr.analyses.decompiler.structured_codegen.c.CContinue method*), 672
__init__() (*angr.analyses.decompiler.structured_codegen.c.CDirtyExpression method*), 676
__init__() (*angr.analyses.decompiler.structured_codegen.c.CDirtyStatement method*), 683
__init__() (*angr.analyses.decompiler.structured_codegen.c.CDoWhileLoop method*), 678
__init__() (*angr.analyses.decompiler.structured_codegen.c.CExpression method*), 674
__init__() (*angr.analyses.decompiler.structured_codegen.c.CExpressionStatement method*), 673
__init__() (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable method*), 676
__init__() (*angr.analyses.decompiler.structured_codegen.c.CForLoop method*), 679
__init__() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 675
__init__() (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall method*), 672
__init__() (*angr.analyses.decompiler.structured_codegen.c.CGoto method*), 677
__init__() (*angr.analyses.decompiler.structured_codegen.c.CITE method*), 677
__init__() (*angr.analyses.decompiler.structured_codegen.c.CIfBreak method*), 682
__init__() (*angr.analyses.decompiler.structured_codegen.c.CIfElse method*), 675
__init__() (*angr.analyses.decompiler.structured_codegen.c.CIncompleteStatement method*), 675
__init__() (*angr.analyses.decompiler.structured_codegen.c.CIndexedVariable method*), 676
__init__() (*angr.analyses.decompiler.structured_codegen.c.CLabel method*), 679
__init__() (*angr.analyses.decompiler.structured_codegen.c.CMultiStatement method*), 678
__init__() (*angr.analyses.decompiler.structured_codegen.c.CRegister method*), 682
__init__() (*angr.analyses.decompiler.structured_codegen.c.CReturn method*), 682
__init__() (*angr.analyses.decompiler.structured_codegen.c.CStatement method*), 677
__init__() (*angr.analyses.decompiler.structured_codegen.c.CStatements method*), 673
__init__() (*angr.analyses.decompiler.structured_codegen.c.CStructField method*), 674
__init__() (*angr.analyses.decompiler.structured_codegen.c.CStructFieldRef method*), 678
__init__() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWriter method*), 684
__init__() (*angr.analyses.decompiler.structured_codegen.c.CSwitchCase method*), 684

method), 676
__init__() (angr.analyses.decompiler.structured_codegen.c.CTypeCast method), 681
__init__() (angr.analyses.decompiler.structured_codegen.c.CUnaryOp method), 680
__init__() (angr.analyses.decompiler.structured_codegen.c.CUnsupportedStatement method), 678
__init__() (angr.analyses.decompiler.structured_codegen.c.CVEXCallExpression method), 683
__init__() (angr.analyses.decompiler.structured_codegen.c.CVariable method), 679
__init__() (angr.analyses.decompiler.structured_codegen.c.CVariableField method), 680
__init__() (angr.analyses.decompiler.structured_codegen.c.CVectorConvert method), 683
__init__() (angr.analyses.decompiler.structured_codegen.c.CWhileLoop method), 674
__init__() (angr.analyses.decompiler.structured_codegen.c_initialize.PtrueCanalyst method), 689
__init__() (angr.analyses.decompiler.structured_codegen.c_initialize.SeanlizeCanalyst method), 688
__init__() (angr.analyses.decompiler.structured_codegen.dummy.DummyStructuredCodeGenerator method), 691
__init__() (angr.analyses.decompiler.structured_codegen.dwarf_import.ImportSourceCode method), 691
__init__() (angr.analyses.decompiler.structured_codegen.dwarf_import.ImportedUse method), 691
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustAllBlock method), 694
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustArray method), 700
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustArrayTypeLength method), 706
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustAssignment method), 697
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustBinaryOp method), 703
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustBreak method), 695
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustClosingObject method), 706
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustConstant method), 703
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustConstant method), 692
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustContinue method), 696
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustDirtyExpression method), 706
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustDirtyStatement method), 699
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustDoWhileLoop method), 694
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustEnum method), 700
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustExpre method), 693
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustFakeV method), 701
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustForLo method), 695
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustFunct method), 692
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustFunct method), 697
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustFunct method), 697
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustGoto method), 698
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustITE method), 704
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustIfBrea method), 695
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustIfElse method), 695
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustIfLet method), 696
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustIncom method), 699
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustIndex method), 702
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustInfinit method), 694
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustLabel method), 699
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustLet method), 700
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustMultiS method), 705
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustPatter method), 696
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustRegist method), 704
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustReturn method), 698
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustStatem method), 693
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustString method), 701
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustStruct method), 699
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustStruct method), 700
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustStruct method), 706
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustStruct method), 706
__init__() (angr.analyses.decompiler.structured_codegen.rust.RustSwitch

*method*), 696
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustTypeCast method*), 703
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustUnaryOp method*), 702
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustUnsupportedSyntax method*), 698
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustVEXCallExpression method*), 705
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustVariable method*), 701
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustVariableField method*), 702
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustVexConvert method*), 705
__init__() (*angr.analyses.decompiler.structured_codegen.rust_like.RustWhileLoop method*), 694
__init__() (*angr.analyses.decompiler.structurer_nodes.BaseNode method*), 710
__init__() (*angr.analyses.decompiler.structurer_nodes.BreakNode method*), 712
__init__() (*angr.analyses.decompiler.structurer_nodes.CascadingConditionNode method*), 712
__init__() (*angr.analyses.decompiler.structurer_nodes.CodeNode method*), 711
__init__() (*angr.analyses.decompiler.structurer_nodes.ConditionNode method*), 711
__init__() (*angr.analyses.decompiler.structurer_nodes.ConditionalBreakNode method*), 713
__init__() (*angr.analyses.decompiler.structurer_nodes.ContinueNode method*), 713
__init__() (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseNode method*), 714
__init__() (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseNode method*), 713
__init__() (*angr.analyses.decompiler.structurer_nodes.LoopNode method*), 712
__init__() (*angr.analyses.decompiler.structurer_nodes.MultiNode method*), 710
__init__() (*angr.analyses.decompiler.structurer_nodes.SequenceNode method*), 711
__init__() (*angr.analyses.decompiler.structurer_nodes.SwitchCaseNode method*), 713
__init__() (*angr.analyses.decompiler.structuring.dream.DreamStructurer method*), 716
__init__() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer method*), 717
__init__() (*angr.analyses.decompiler.structuring.recursive_structurer.RecursiveStructurer method*), 718
__init__() (*angr.analyses.decompiler.structuring.sailr.SAILRStructurer method*), 719
__init__() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase method*), 719
__init__() (*angr.analyses.decompiler.variable_map.VariableMap method*), 727
__init__() (*angr.analyses.deobfuscator.api_obf_finder.APIDeobFuncDes method*), 734
__init__() (*angr.analyses.deobfuscator.api_obf_finder.APIObfuscationFi method*), 735
__init__() (*angr.analyses.deobfuscator.api_obf_finder.Type1AssignmentI method*), 734
__init__() (*angr.analyses.deobfuscator.api_obf_type2_finder.APIObfusca method*), 736
__init__() (*angr.analyses.deobfuscator.api_obf_type2_finder.APIObfusca method*), 736
__init__() (*angr.analyses.deobfuscator.data_transformation_embedder.D method*), 737
__init__() (*angr.analyses.deobfuscator.hash_lookup_api_deobfuscator.F method*), 737
__init__() (*angr.analyses.deobfuscator.hash_lookup_api_deobfuscator.H method*), 737
__init__() (*angr.analyses.deobfuscator.irsb_reg_collector.IRSBRegisterC method*), 737
__init__() (*angr.analyses.deobfuscator.scope_ops_analyzer.ScopeOpsAn method*), 738
__init__() (*angr.analyses.deobfuscator.scope_ops_analyzer.ScopeOpsWa method*), 738
__init__() (*angr.analyses.deobfuscator.string_obf_finder.StringDeobFunc method*), 738
__init__() (*angr.analyses.deobfuscator.string_obf_finder.StringObfuscati method*), 738
__init__() (*angr.analyses.deobfuscator.string_obf_opt_passes.StringObf method*), 739
__init__() (*angr.analyses.disassembly.BlockStart method*), 740
__init__() (*angr.analyses.disassembly.HeadStatement method*), 742
__init__() (*angr.analyses.disassembly.Disassembly method*), 743
__init__() (*angr.analyses.disassembly.FuncComment method*), 742
__init__() (*angr.analyses.disassembly.FunctionStart method*), 740
__init__() (*angr.analyses.disassembly.Hook method*), 740
__init__() (*angr.analyses.disassembly.IROp method*), 740
__init__() (*angr.analyses.disassembly.Instruction method*), 740
__init__() (*angr.analyses.disassembly.Label method*), 740
__init__() (*angr.analyses.disassembly.MemoryOperand method*), 742
__init__() (*angr.analyses.disassembly.Opcode method*), 741
__init__() (*angr.analyses.disassembly.Operand method*), 741
__init__() (*angr.analyses.disassembly.Register*

*method*), 742
__init__() (*angr.analyses.disassembly.SootExpression method*), 741
__init__() (*angr.analyses.disassembly.SootExpressionInvoke method*), 741
__init__() (*angr.analyses.disassembly.SootExpressionStaticFieldRef method*), 741
__init__() (*angr.analyses.disassembly.SootExpressionTarget method*), 741
__init__() (*angr.analyses.disassembly.SootStatement method*), 741
__init__() (*angr.analyses.disassembly.Value method*), 742
__init__() (*angr.analyses.dominance_frontier.DominanceFrontier method*), 743
__init__() (*angr.analyses.fcp.fcp.FCPState method*), 745
__init__() (*angr.analyses.fcp.fcp.FastConstantPropagation method*), 746
__init__() (*angr.analyses.fcp.fcp.SV method*), 744
__init__() (*angr.analyses.fcp.fcp.SimEngineFCPVEX method*), 746
__init__() (*angr.analyses.find_objects_static.PossibleObject method*), 746
__init__() (*angr.analyses.find_objects_static.StaticObjectFinder method*), 747
__init__() (*angr.analyses.flirt.flirt.FlirtAnalysis method*), 755
__init__() (*angr.analyses.flirt.flirt_function.FlirtFunction method*), 756
__init__() (*angr.analyses.flirt.flirt_matcher.FlirtMatcher method*), 756
__init__() (*angr.analyses.flirt.flirt_module.FlirtModule method*), 757
__init__() (*angr.analyses.flirt.flirt_node.FlirtNode method*), 757
__init__() (*angr.analyses.flirt.flirt_sig.FlirtSignature method*), 758
__init__() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed method*), 758
__init__() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 763
__init__() (*angr.analyses.forward_analysis.job_info.JobInfo method*), 764
__init__() (*angr.analyses.forward_analysis.visitors.call_graph.CallGraphVisitor method*), 766
__init__() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method*), 766
__init__() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 766
__init__() (*angr.analyses.forward_analysis.visitors.loop.LoopVisitor method*), 768
__init__() (*angr.analyses.forward_analysis.visitors.single_node_graph.SingleNodeGraphVisitor method*), 769
__init__() (*angr.analyses.identifier.custom_callable.IdentifierCallable method*), 770
__init__() (*angr.analyses.identifier.func.Func method*), 771
__init__() (*angr.analyses.identifier.func.TestData method*), 771
__init__() (*angr.analyses.identifier.functions.atoi.atoi method*), 772
__init__() (*angr.analyses.identifier.functions.based_atoi.based_atoi method*), 772
__init__() (*angr.analyses.identifier.functions.fdprintf.fdprintf method*), 773
__init__() (*angr.analyses.identifier.functions.free.free method*), 773
__init__() (*angr.analyses.identifier.functions.int2str.int2str method*), 774
__init__() (*angr.analyses.identifier.functions.int2str.int2str_v2 method*), 774
__init__() (*angr.analyses.identifier.functions.int2str.int2str_v3 method*), 774
__init__() (*angr.analyses.identifier.functions.int2str.int2str_v4 method*), 774
__init__() (*angr.analyses.identifier.functions.malloc.malloc method*), 774
__init__() (*angr.analyses.identifier.functions.memcmp.memcmp method*), 775
__init__() (*angr.analyses.identifier.functions.memcpy.memcpy method*), 775
__init__() (*angr.analyses.identifier.functions.memset.memset method*), 776
__init__() (*angr.analyses.identifier.functions.printf.printf method*), 777
__init__() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 777
__init__() (*angr.analyses.identifier.functions.recv_until.receive_until_fd method*), 777
__init__() (*angr.analyses.identifier.functions.skip_calloc.calloc method*), 778
__init__() (*angr.analyses.identifier.functions.skip_realloc.realloc method*), 778
__init__() (*angr.analyses.identifier.functions.skip_recv_n.receive_n2 method*), 779
__init__() (*angr.analyses.identifier.functions.skip_recv_n.receive_n3 method*), 779
__init__() (*angr.analyses.identifier.functions.skip_recv_n.receive_n4 method*), 778
__init__() (*angr.analyses.identifier.functions.snprintf.snprintf method*), 779
__init__() (*angr.analyses.identifier.functions.sprintf.sprintf method*), 780
__init__() (*angr.analyses.identifier.functions.strcasecmp.strcasecmp method*), 781
__init__() (*angr.analyses.identifier.functions.strcmp.strcmp method*), 781

__init__() (angr.analyses.identifier.functions.strcpy.strcpy method), 782
__init__() (angr.analyses.identifier.functions.strncmp.strncmp method), 783
__init__() (angr.analyses.identifier.functions.strncpy.strncpy method), 784
__init__() (angr.analyses.identifier.functions.strtol.strtol method), 784
__init__() (angr.analyses.identifier.identify.FuncInfo method), 784
__init__() (angr.analyses.identifier.identify.Identifier method), 785
__init__() (angr.analyses.identifier.runner.Runner method), 785
__init__() (angr.analyses.init_finder.SimEngineInitFinderVEX method), 786
__init__() (angr.analyses.language_detector.LanguageDetectionResult method), 786
__init__() (angr.analyses.language_detector.LanguageDetector method), 787
__init__() (angr.analyses.loop_analysis.loop_analysis.AssignmentNode method), 788
__init__() (angr.analyses.loop_analysis.loop_analysis.BinOpNode method), 788
__init__() (angr.analyses.loop_analysis.loop_analysis.ConstNode method), 788
__init__() (angr.analyses.loop_analysis.loop_analysis.LoopAnalysis method), 790
__init__() (angr.analyses.loop_analysis.loop_analysis.LoopBlockAssignmentCollector method), 789
__init__() (angr.analyses.loop_analysis.loop_analysis.LoopBlockConditionCollector method), 789
__init__() (angr.analyses.loop_analysis.loop_analysis.LoopVisitor method), 790
__init__() (angr.analyses.loop_analysis.loop_analysis.VarNode method), 788
__init__() (angr.analyses.loop_unroller.loop_unroller.LoopUnroller method), 791
__init__() (angr.analyses.loopfinder.Loop method), 791
__init__() (angr.analyses.loopfinder.LoopFinder method), 791
__init__() (angr.analyses.outliner.outliner.Outliner method), 792
__init__() (angr.analyses.patchfinder.AtypicallyAlignedFunction method), 793
__init__() (angr.analyses.patchfinder.FunctionAlignmentAnalysis method), 793
__init__() (angr.analyses.patchfinder.OverlappingFunctionsAnalysis method), 792
__init__() (angr.analyses.patchfinder.PatchFinderAnalysis method), 793
__init__() (angr.analyses.patchfinder.PatchedOutFunctionality method), 793
__init__() (angr.analyses.pathfinder.Pathfinder method), 795
__init__() (angr.analyses.pathfinder.SimStateMarker method), 794
__init__() (angr.analyses.pathfinder.TestPathReport method), 794
__init__() (angr.analyses.propagator.engine_base.SimEnginePropagator method), 796
__init__() (angr.analyses.propagator.values.Top method), 798
__init__() (angr.analyses.propagator.vex_vars.VEXMemVar method), 798
__init__() (angr.analyses.propagator.vex_vars.VEXReg method), 798
__init__() (angr.analyses.propagator.vex_vars.VEXTmp method), 798
__init__() (angr.analyses.proximity_graph.BaseProxiNode method), 799
__init__() (angr.analyses.proximity_graph.CallProxiNode method), 799
__init__() (angr.analyses.proximity_graph.FunctionProxiNode method), 799
__init__() (angr.analyses.proximity_graph.IntegerProxiNode method), 800
__init__() (angr.analyses.proximity_graph.ProximityGraphAnalysis method), 800
__init__() (angr.analyses.proximity_graph.StringProxiNode method), 799
__init__() (angr.analyses.proximity_graph.UnknownProxiNode method), 800
__init__() (angr.analyses.proximity_graph.VariableProxiNode method), 799
__init__() (angr.analyses.purity.engine.DataSource method), 801
__init__() (angr.analyses.purity.engine.DataUsage method), 801
__init__() (angr.analyses.purity.engine.PurityEngineAIL method), 803
__init__() (angr.analyses.purity.engine.ResultType method), 803
__init__() (angr.analyses.purity.engine.StateType method), 802
__init__() (angr.analyses.reaching_definitions.call_trace.CallSite method), 834
__init__() (angr.analyses.reaching_definitions.call_trace.CallTrace method), 834
__init__() (angr.analyses.reaching_definitions.dep_graph.DepGraph method), 835
__init__() (angr.analyses.reaching_definitions.engine_ail.SimEngineRDA method), 838
__init__() (angr.analyses.reaching_definitions.engine_vex.SimEngineRD method), 838
__init__() (angr.analyses.reaching_definitions.function_handler.Function method), 841

__init__() (angr.analyses.reaching_definitions.function_handler.FunctionCallDataView method), 842
__init__() (angr.analyses.reaching_definitions.function_handler.FunctionCallRelationships method), 842
__init__() (angr.analyses.reaching_definitions.function_handler.FunctionEffect method), 839
__init__() (angr.analyses.reaching_definitions.function_handler.FunctionHandler method), 843
__init__() (angr.analyses.reaching_definitions.heap_allocator.HeapAllocator method), 857
__init__() (angr.analyses.reaching_definitions.rd_initializer.RDAStateInitializer method), 857
__init__() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method), 860
__init__() (angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method), 867
__init__() (angr.analyses.reaching_definitions.subject.Subject method), 869
__init__() (angr.analyses.reassembler.BasicBlock method), 872
__init__() (angr.analyses.reassembler.Data method), 874
__init__() (angr.analyses.reassembler.DataLabel method), 870
__init__() (angr.analyses.reassembler.FunctionLabel method), 871
__init__() (angr.analyses.reassembler.Instruction method), 872
__init__() (angr.analyses.reassembler.Label method), 870
__init__() (angr.analyses.reassembler.NotypeLabel method), 871
__init__() (angr.analyses.reassembler.ObjectLabel method), 871
__init__() (angr.analyses.reassembler.Operand method), 871
__init__() (angr.analyses.reassembler.Procedure method), 873
__init__() (angr.analyses.reassembler.ProcedureChunk method), 874
__init__() (angr.analyses.reassembler.Reassembler method), 875
__init__() (angr.analyses.reassembler.Relocation method), 875
__init__() (angr.analyses.reassembler.SymbolManager method), 871
__init__() (angr.analyses.s_liveness.SLivenessAnalysis method), 878
__init__() (angr.analyses.s_liveness.SLivenessModel method), 878
__init__() (angr.analyses.s_propagator.SPropagator method), 879
__init__() (angr.analyses.s_propagator.SPropagatorAnalysis method), 880
__init__() (angr.analyses.s_propagator.SPropagatorModel method), 879
__init__() (angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method), 885
__init__() (angr.analyses.s_reaching_definitions.s_rda_view.RegVVarProvider method), 889
__init__() (angr.analyses.s_reaching_definitions.s_rda_view.SRDAView method), 889
__init__() (angr.analyses.s_reaching_definitions.s_rda_view.StackVVarProvider method), 889
__init__() (angr.analyses.s_reaching_definitions.s_reaching_definitions.SReachingDefinitionsAnalysis method), 891
__init__() (angr.analyses.s_reaching_definitions.s_reaching_definitions.SReachingDefinitionsState method), 891
__init__() (angr.analyses.self_modifying_code.SelfModifyingCodeAnalysis method), 892
__init__() (angr.analyses.smc.TraceClassifier method), 891
__init__() (angr.analyses.soot_class_hierarchy.SootClassHierarchy method), 893
__init__() (angr.analyses.stack_pointer_tracker.Constant method), 893
__init__() (angr.analyses.stack_pointer_tracker.Eq method), 894
__init__() (angr.analyses.stack_pointer_tracker.FrozenStackPointerTracker method), 894
__init__() (angr.analyses.stack_pointer_tracker.OffsetVal method), 894
__init__() (angr.analyses.stack_pointer_tracker.Register method), 894
__init__() (angr.analyses.stack_pointer_tracker.StackPointerTrackerState method), 895
__init__() (angr.analyses.static_hooker.StaticHooker method), 896
__init__() (angr.analyses.typehoon.simple_solver.ConstraintGraphNode method), 901
__init__() (angr.analyses.typehoon.simple_solver.ConstraintGraphTag method), 901
__init__() (angr.analyses.typehoon.simple_solver.FORGOTTEN method), 901
__init__() (angr.analyses.typehoon.simple_solver.RecursiveRefNode method), 900
__init__() (angr.analyses.typehoon.simple_solver.SimpleSolver method), 902
__init__() (angr.analyses.typehoon.simple_solver.Sketch method), 900
__init__() (angr.analyses.typehoon.simple_solver.SketchNode method), 900
__init__() (angr.analyses.typehoon.simple_solver.TypeLattice method), 899
__init__() (angr.analyses.typehoon.translator.TypeTranslator method), 905
__init__() (angr.analyses.typehoon.typeconsts.Array method), 910

__init__() (angr.analyses.typehoon.typeconsts.Enum method), 912
__init__() (angr.analyses.typehoon.typeconsts.EnumVariant method), 911
__init__() (angr.analyses.typehoon.typeconsts.Function method), 912
__init__() (angr.analyses.typehoon.typeconsts.IntVar method), 909
__init__() (angr.analyses.typehoon.typeconsts.Pointer method), 909
__init__() (angr.analyses.typehoon.typeconsts.Pointer32 method), 910
__init__() (angr.analyses.typehoon.typeconsts.Pointer64 method), 910
__init__() (angr.analyses.typehoon.typeconsts.RustEnum method), 911
__init__() (angr.analyses.typehoon.typeconsts.Struct method), 911
__init__() (angr.analyses.typehoon.typeconsts.TypeConstant method), 907
__init__() (angr.analyses.typehoon.typeconsts.TypeVariableReference method), 913
__init__() (angr.analyses.typehoon.typehoon.Typehoon method), 913
__init__() (angr.analyses.typehoon.typevars.Add method), 915
__init__() (angr.analyses.typehoon.typevars.AddN method), 919
__init__() (angr.analyses.typehoon.typevars.BaseLabel method), 919
__init__() (angr.analyses.typehoon.typevars.ConvertTo method), 920
__init__() (angr.analyses.typehoon.typevars.DerivedTypeVariable method), 917
__init__() (angr.analyses.typehoon.typevars.Equivalence method), 915
__init__() (angr.analyses.typehoon.typevars.Existence method), 915
__init__() (angr.analyses.typehoon.typevars.FuncIn method), 919
__init__() (angr.analyses.typehoon.typevars.FuncOut method), 919
__init__() (angr.analyses.typehoon.typevars.HasField method), 920
__init__() (angr.analyses.typehoon.typevars.ReinterpretAs method), 920
__init__() (angr.analyses.typehoon.typevars.Sub method), 916
__init__() (angr.analyses.typehoon.typevars.SubN method), 919
__init__() (angr.analyses.typehoon.typevars.Subtype method), 915
__init__() (angr.analyses.typehoon.typevars.TypeVariable method), 917
__init__() (angr.analyses.typehoon.typevars.TypeVariableManager method), 916
__init__() (angr.analyses.typehoon.typevars.TypeVariables method), 918
__init__() (angr.analyses.typehoon.variance.Variance method), 920
__init__() (angr.analyses.unpacker.obfuscation_detector.ObfuscationDet method), 921
__init__() (angr.analyses.unpacker.packing_detector.PackingDetector method), 922
__init__() (angr.analyses.variable_recovery.annotations.StackLocationA method), 923
__init__() (angr.analyses.variable_recovery.annotations.VariableSource method), 923
__init__() (angr.analyses.variable_recovery.engine_ail.SimEngineVRAIL method), 923
__init__() (angr.analyses.variable_recovery.engine_base.RichR method), 924
__init__() (angr.analyses.variable_recovery.engine_base.SimEngineVRB method), 924
__init__() (angr.analyses.variable_recovery.engine_vex.SimEngineVRVE method), 925
__init__() (angr.analyses.variable_recovery.irsb_scanner.VEXIRSBScan method), 925
__init__() (angr.analyses.variable_recovery.variable_recovery.VariableR method), 926
__init__() (angr.analyses.variable_recovery.variable_recovery.VariableR method), 925
__init__() (angr.analyses.variable_recovery.variable_recovery_base.Var method), 927
__init__() (angr.analyses.variable_recovery.variable_recovery_base.Var method), 927
__init__() (angr.analyses.variable_recovery.variable_recovery_base.Var method), 927
__init__() (angr.analyses.variable_recovery.variable_recovery_fast.Varia method), 930
__init__() (angr.analyses.veritesting.CallTracingFilter method), 931
__init__() (angr.analyses.veritesting.Veritesting method), 931
__init__() (angr.analyses.vfg.AnalysisTask method), 933
__init__() (angr.analyses.vfg.CallAnalysis method), 933
__init__() (angr.analyses.vfg.FunctionAnalysis method), 933
__init__() (angr.analyses.vfg.PendingJob method), 932
__init__() (angr.analyses.vfg.VFG method), 935
__init__() (angr.analyses.vfg.VFGJob method), 932
__init__() (angr.analyses.vfg.VFGNode method), 934
__init__() (angr.analyses.vsa_ddg.DefUseChain method), 936
__init__() (angr.analyses.vsa_ddg.VSA_DDG

method), 936
__init__() (angr.analyses.vtable.Vtable method), 937
__init__() (angr.analyses.vtable.VtableFinder method), 937
__init__() (angr.analyses.xrefs.SimEngineXRefsVEX method), 937
__init__() (angr.angrdb.db.AngrDB method), 940
__init__() (angr.annocfg.AnnotatedCFG method), 957
__init__() (angr.blade.Blade method), 958
__init__() (angr.block.Block method), 960
__init__() (angr.block.CapstoneInsn method), 960
__init__() (angr.block.DisassemblerBlock method), 959
__init__() (angr.block.PCodeInsn method), 960
__init__() (angr.block.SootBlock method), 961
__init__() (angr.callable.Callable method), 961
__init__() (angr.calling_conventions.AllocHelper method), 962
__init__() (angr.calling_conventions.ArgSession method), 966
__init__() (angr.calling_conventions.PointerWrapper method), 962
__init__() (angr.calling_conventions.SerializableCounter method), 963
__init__() (angr.calling_conventions.SerializableListIterator method), 963
__init__() (angr.calling_conventions.SimArrayArg method), 966
__init__() (angr.calling_conventions.SimCC method), 967
__init__() (angr.calling_conventions.SimCC.ArgSession method), 968
__init__() (angr.calling_conventions.SimComboArg method), 965
__init__() (angr.calling_conventions.SimFunctionArgument method), 964
__init__() (angr.calling_conventions.SimLyingRegArg method), 971
__init__() (angr.calling_conventions.SimReferenceArgument method), 966
__init__() (angr.calling_conventions.SimRegArg method), 964
__init__() (angr.calling_conventions.SimStackArg method), 965
__init__() (angr.calling_conventions.SimStructArg method), 965
__init__() (angr.calling_conventions.UsercallArgSession method), 967
__init__() (angr.code_location.AILCodeLocation method), 986
__init__() (angr.code_location.CodeLocation method), 985
__init__() (angr.codenode.BlockNode method), 987
__init__() (angr.codenode.CodeNode method), 987
__init__() (angr.codenode.FuncNode method), 988
__init__() (angr.codenode.HookNode method), 988
__init__() (angr.codenode.SootBlockNode method), 987
__init__() (angr.concretization_strategies.base.SimConcretizationStrateg method), 991
__init__() (angr.distributed.server.Server method), 994
__init__() (angr.distributed.worker.BadStatesDropper method), 994
__init__() (angr.distributed.worker.ExplorationStatusNotifier method), 994
__init__() (angr.distributed.worker.Worker method), 995
__init__() (angr.emulator.Emulator method), 995
__init__() (angr.engines.ail.callstack.AILCallStack method), 1003
__init__() (angr.engines.ail.engine_light.SimEngineAILSimState method), 1003
__init__() (angr.engines.engine.SimEngine method), 1004
__init__() (angr.engines.light.data.ArithmeticExpression method), 1010
__init__() (angr.engines.light.data.RegisterOffset method), 1010
__init__() (angr.engines.light.data.SpOffset method), 1010
__init__() (angr.engines.light.engine.IRTop method), 1010
__init__() (angr.engines.light.engine.SimEngineLight method), 1011
__init__() (angr.engines.light.engine.SimEngineLightAIL method), 1013
__init__() (angr.engines.light.engine.SimEngineLightVEX method), 1013
__init__() (angr.engines.pcode.behavior.BehaviorFactory method), 1034
__init__() (angr.engines.pcode.behavior.OpBehavior method), 1014
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolAnd method), 1028
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolNegate method), 1027
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolOr method), 1028
__init__() (angr.engines.pcode.behavior.OpBehaviorBoolXor method), 1027
__init__() (angr.engines.pcode.behavior.OpBehaviorCopy method), 1016
__init__() (angr.engines.pcode.behavior.OpBehaviorEqual method), 1016
__init__() (angr.engines.pcode.behavior.OpBehaviorFloatAbs method), 1031
__init__() (angr.engines.pcode.behavior.OpBehaviorFloatAdd

*method*), 1030
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatCeil* *method*), 1032
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatDiv* *method*), 1030
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatEqual* *method*), 1029
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatFloat2Float* *method*), 1031
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatFloor* *method*), 1032
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatInt2Float* *method*), 1031
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatLess* *method*), 1029
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatLessEqual* *method*), 1029
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatMult* *method*), 1030
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNan* *method*), 1030
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNeg* *method*), 1031
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatNotEqual* *method*), 1029
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatRound* *method*), 1032
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatSqrt* *method*), 1031
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatSub* *method*), 1030
__init__() (*angr.engines.pcode.behavior.OpBehaviorFloatTrunc* *method*), 1032
__init__() (*angr.engines.pcode.behavior.OpBehaviorInt2Comp* *method*), 1021
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntAdd* *method*), 1019
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntAnd* *method*), 1023
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntCarry* *method*), 1020
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntDiv* *method*), 1025
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLeft* *method*), 1024
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLess* *method*), 1018
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual* *method*), 1018
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntMult* *method*), 1025
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntNegate* *method*), 1022
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntOr* *method*), 1023
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntRem* *method*), 1026
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntRight* *method*), 1024
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSborrow* *method*), 1021
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntScarry* *method*), 1021
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSdiv* *method*), 1026
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSext* *method*), 1019
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSless* *method*), 1017
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual* *method*), 1017
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSrem* *method*), 1027
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSright* *method*), 1024
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntSub* *method*), 1020
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntXor* *method*), 1022
__init__() (*angr.engines.pcode.behavior.OpBehaviorIntZext* *method*), 1019
__init__() (*angr.engines.pcode.behavior.OpBehaviorLzcount* *method*), 1033
__init__() (*angr.engines.pcode.behavior.OpBehaviorNotEqual* *method*), 1016
__init__() (*angr.engines.pcode.behavior.OpBehaviorPiece* *method*), 1032
__init__() (*angr.engines.pcode.behavior.OpBehaviorPopcount* *method*), 1033
__init__() (*angr.engines.pcode.behavior.OpBehaviorSubpiece* *method*), 1033
__init__() (*angr.engines.pcode.emulate.PcodeEmulatorMixin* *method*), 1036
__init__() (*angr.engines.pcode.engine.HeavyPcodeMixin* *method*), 1037
__init__() (*angr.engines.pcode.lifter.ExitStatement* *method*), 1037
__init__() (*angr.engines.pcode.lifter.IRSB* *method*), 1038
__init__() (*angr.engines.pcode.lifter.Lifter* *method*), 1041
__init__() (*angr.engines.pcode.lifter.PcodeBasicBlockLifter* *method*), 1043
__init__() (*angr.engines.pcode.lifter.PcodeDisassemblerInsn* *method*), 1037
__init__() (*angr.engines.pcode.lifter.PcodeLifter* *method*), 1043
__init__() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin*

*method*), 1044
__init__() (*angr.engines.soot.expressions.base.SimSootExpr method*), 1050
__init__() (*angr.engines.soot.expressions.invoke.InvokeBase method*), 1051
__init__() (*angr.engines.soot.statements.base.SimSootStmt method*), 1054
__init__() (*angr.engines.soot.statements.return_.SimSootStmt_Return method*), 1054
__init__() (*angr.engines.soot.statements.return_.SimSootStmt_ReturnVoid method*), 1055
__init__() (*angr.engines.soot.values.arrayref.SimSootValue_ArrayBaseRef method*), 1059
__init__() (*angr.engines.soot.values.arrayref.SimSootValue_ArrayRef method*), 1059
__init__() (*angr.engines.soot.values.constants.SimSootValue_IntConstant method*), 1060
__init__() (*angr.engines.soot.values.instancefieldref.SimSootValue_InstanceFieldRef method*), 1060
__init__() (*angr.engines.soot.values.local.SimSootValue_Local method*), 1060
__init__() (*angr.engines.soot.values.paramref.SimSootValue_ParamRef method*), 1061
__init__() (*angr.engines.soot.values.staticfieldref.SimSootValue_StaticFieldRef method*), 1061
__init__() (*angr.engines.soot.values.strref.SimSootValue_StringRef method*), 1061
__init__() (*angr.engines.soot.values.thisref.SimSootValue_ThisRef method*), 1062
__init__() (*angr.engines.successors.SimSuccessors method*), 1063
__init__() (*angr.engines.successors.SuccessorsEngine method*), 1064
__init__() (*angr.engines.unicorn.SimEngineUnicorn method*), 1065
__init__() (*angr.engines.vex.claripy.irop.SimIROp method*), 1074
__init__() (*angr.engines.vex.heavy.actions.TrackActionsMixin method*), 1075
__init__() (*angr.engines.vex.lifter.VEXLifter method*), 1079
__init__() (*angr.engines.vex.light.light.VEXMixin method*), 1080
__init__() (*angr.engines.vex.light.slicing.VEXSlicingMixin method*), 1081
__init__() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1097
__init__() (*angr.exploration_techniques.dfs.DFS method*), 1099
__init__() (*angr.exploration_techniques.director.BaseGoal method*), 1099
__init__() (*angr.exploration_techniques.director.CallFunctionGoal method*), 1100
__init__() (*angr.exploration_techniques.director.Director method*), 1101
__init__() (*angr.exploration_techniques.director.ExecuteAddressGoal method*), 1099
__init__() (*angr.exploration_techniques.driller_core.DrillerCore method*), 1101
__init__() (*angr.exploration_techniques.explorer.Explorer method*), 1102
__init__() (*angr.exploration_techniques.lengthlimiter.LengthLimiter method*), 1102
__init__() (*angr.exploration_techniques.local_loop_seer.LocalLoopSeer method*), 1102
__init__() (*angr.exploration_techniques.loop_seer.LoopSeer method*), 1102
__init__() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 1103
__init__() (*angr.exploration_techniques.memory_watcher.MemoryWatcher method*), 1103
__init__() (*angr.exploration_techniques.oppologist.Oppologist method*), 1104
__init__() (*angr.exploration_techniques.slicecutor.Slicecutor method*), 1104
__init__() (*angr.exploration_techniques.spiller.PickledStatesDb method*), 1105
__init__() (*angr.exploration_techniques.spiller.PickledStatesList method*), 1104
__init__() (*angr.exploration_techniques.spiller.Spiller method*), 1105
__init__() (*angr.exploration_techniques.stochastic.StochasticSearch method*), 1106
__init__() (*angr.exploration_techniques.suggestions.Suggestions method*), 1106
__init__() (*angr.exploration_techniques.tech_builder.TechniqueBuilder method*), 1107
__init__() (*angr.exploration_techniques.threading.Threading method*), 1107
__init__() (*angr.exploration_techniques.timeout.Timeout method*), 1107
__init__() (*angr.exploration_techniques.tracer.RepHook method*), 1108
__init__() (*angr.exploration_techniques.tracer.Tracer method*), 1108
__init__() (*angr.exploration_techniques.unique.UniqueSearch method*), 1109
__init__() (*angr.exploration_techniques.veritesting.Veritesting method*), 1109
__init__() (*angr.factory.AngrObjectFactory method*), 1109
__init__() (*angr.keyed_region.KeyedRegion method*), 1116
__init__() (*angr.keyed_region.RegionObject method*), 1116
__init__() (*angr.keyed_region.StoredObject method*), 1115
__init__() (*angr.knowledge_base.KnowledgeBase*

method), 1119
__init__() (angr.knowledge_plugins.bookmarks.Bookmark method), 1149
__init__() (angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method), 1150
__init__() (angr.knowledge_plugins.cfg.block_id.BlockID method), 1165
__init__() (angr.knowledge_plugins.cfg.cfg_manager.CFGManager method), 1165
__init__() (angr.knowledge_plugins.cfg.cfg_model.CFGModel method), 1166
__init__() (angr.knowledge_plugins.cfg.cfg_node.CFGNode method), 1173
__init__() (angr.knowledge_plugins.cfg.cfg_node.CFGNodeCreationInfo method), 1173
__init__() (angr.knowledge_plugins.cfg.indirect_jump.IndirectJump method), 1176
__init__() (angr.knowledge_plugins.cfg.indirect_jump.JumpTableInfo method), 1175
__init__() (angr.knowledge_plugins.cfg.memory_data.MemoryData method), 1178
__init__() (angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method), 1182
__init__() (angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method), 1179
__init__() (angr.knowledge_plugins.cfg.spilling_digraph.DiffDict method), 1187
__init__() (angr.knowledge_plugins.cfg.spilling_digraph.SpillingAdjDict method), 1187
__init__() (angr.knowledge_plugins.comments.Comment method), 1189
__init__() (angr.knowledge_plugins.custom_strings.CustomStrings method), 1191
__init__() (angr.knowledge_plugins.debug_variables.DebugVariable method), 1192
__init__() (angr.knowledge_plugins.debug_variables.DebugVariableContainer method), 1192
__init__() (angr.knowledge_plugins.debug_variables.DebugVariableManager method), 1192
__init__() (angr.knowledge_plugins.functions.function.Function method), 1210
__init__() (angr.knowledge_plugins.functions.function.FunctionInfo method), 1209
__init__() (angr.knowledge_plugins.functions.function_manager.FunctionDict method), 1218
__init__() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 1221
__init__() (angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict method), 1219
__init__() (angr.knowledge_plugins.functions.soot_function.SootFunction method), 1228
__init__() (angr.knowledge_plugins.key_definitions.atoms.Atom method), 1242
__init__() (angr.knowledge_plugins.key_definitions.atoms.MemoryLocation method), 1245
__init__() (angr.knowledge_plugins.key_definitions.definition.Definition method), 1247
__init__() (angr.knowledge_plugins.key_definitions.definition.DefinitionL method), 1246
__init__() (angr.knowledge_plugins.key_definitions.environment.Environ method), 1247
__init__() (angr.knowledge_plugins.key_definitions.heap_address.HeapA method), 1248
__init__() (angr.knowledge_plugins.key_definitions.key_definition_mana method), 1249
__init__() (angr.knowledge_plugins.key_definitions.key_definition_mana method), 1248
__init__() (angr.knowledge_plugins.key_definitions.live_definitions.Defin method), 1249
__init__() (angr.knowledge_plugins.key_definitions.live_definitions.LiveL method), 1250
__init__() (angr.knowledge_plugins.key_definitions.liveness.Liveness method), 1257
__init__() (angr.knowledge_plugins.key_definitions.rd_model.ReachingD method), 1259
__init__() (angr.knowledge_plugins.key_definitions.tag.FunctionTag method), 1261
__init__() (angr.knowledge_plugins.key_definitions.tag.Tag method), 1261
__init__() (angr.knowledge_plugins.key_definitions.uses.Uses method), 1262
__init__() (angr.knowledge_plugins.labels.Labels method), 1264
__init__() (angr.knowledge_plugins.obfuscations.Obfuscations method), 1264
__init__() (angr.knowledge_plugins.patches.Patch method), 1264
__init__() (angr.knowledge_plugins.patches.PatchManager method), 1265
__init__() (angr.knowledge_plugins.plugin.KnowledgeBasePlugin method), 1266
__init__() (angr.knowledge_plugins.propagations.prop_value.Detail method), 1268
__init__() (angr.knowledge_plugins.propagations.prop_value.PropValue method), 1268
__init__() (angr.knowledge_plugins.propagations.propagation_manager. method), 1270
__init__() (angr.knowledge_plugins.propagations.propagation_model.Pr method), 1271
__init__() (angr.knowledge_plugins.propagations.states.CallExprFinder method), 1272
__init__() (angr.knowledge_plugins.propagations.states.Equivalence method), 1276
__init__() (angr.knowledge_plugins.propagations.states.PropagatorState method), 1272
__init__() (angr.knowledge_plugins.propagations.states.PropagatorVEX method), 1275
__init__() (angr.knowledge_plugins.propagations.states.RegisterAnnotat

method), 1275
__init__() (angr.knowledge_plugins.propagations.states.RegisterCompatibilityAnnotation method), 1275
__init__() (angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method), 1277
__init__() (angr.knowledge_plugins.rtdb.rtdb.RuntimeDbForkCondition method), 1277
__init__() (angr.knowledge_plugins.structured_code.SpillingDecompilationDict method), 1279
__init__() (angr.knowledge_plugins.structured_code.StructuredCodeManager method), 1279
__init__() (angr.knowledge_plugins.types.TypesStore method), 1280
__init__() (angr.knowledge_plugins.variables.spilling_vardict.SpillingVariableInternalDict method), 1282
__init__() (angr.knowledge_plugins.variables.variable_access.VariableAccess method), 1282
__init__() (angr.knowledge_plugins.variables.variable_manager.DecompilationVariableManager method), 1289
__init__() (angr.knowledge_plugins.variables.variable_manager.LiveVariables method), 1283
__init__() (angr.knowledge_plugins.variables.variable_manager.VariableManager method), 1288
__init__() (angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method), 1283
__init__() (angr.knowledge_plugins.xrefs.xref.XRef method), 1291
__init__() (angr.knowledge_plugins.xrefs.xref_manager.XRefManager method), 1292
__init__() (angr.llm_client.LLMClient method), 1293
__init__() (angr.mcp.session.ProjectSession method), 1308
__init__() (angr.mcp.session.SessionManager method), 1308
__init__() (angr.misc.hookset.HookedMethod method), 1316
__init__() (angr.misc.loggers.Loggers method), 1316
__init__() (angr.misc.picklable_lock.PicklableLock method), 1317
__init__() (angr.misc.plugins.PluginHub method), 1317
__init__() (angr.misc.plugins.PluginPreset method), 1319
__init__() (angr.procedures.definitions.SimLibrary method), 1325
__init__() (angr.procedures.definitions.SimSyscallLibrary method), 1330
__init__() (angr.procedures.definitions.SimTypeCollection method), 1324
__init__() (angr.procedures.stubs.format_parser.FormatSpecifier method), 1379
__init__() (angr.procedures.stubs.format_parser.FormatString method), 1378
__init__() (angr.project.Project method), 1389
__init__() (angr.rust.analyses.cleanup_function_identification.CleanupF method), 1406
__init__() (angr.rust.analyses.flirt_sig_propagation.FlirtSigPropagation method), 1407
__init__() (angr.rust.analyses.rust_calling_convention.fact_collector.Ca method), 1408
__init__() (angr.rust.analyses.rust_calling_convention.fact_collector.Ca method), 1408
__init__() (angr.rust.analyses.rust_calling_convention.fact_collector.Co method), 1408
__init__() (angr.rust.analyses.rust_calling_convention.fact_collector.Fac method), 1409
__init__() (angr.rust.analyses.rust_calling_convention.fact_collector.Me method), 1408
__init__() (angr.rust.analyses.rust_calling_convention.pathfinder.Pathfin method), 1409
__init__() (angr.rust.analyses.rust_calling_convention.rust_calling_conv method), 1409
__init__() (angr.rust.analyses.rust_calling_convention.rust_calling_conv method), 1410
__init__() (angr.rust.analyses.rust_symbol_recovery.RustSymbolRecover method), 1410
__init__() (angr.rust.analyses.rust_type_hints.RustTypeHintsAnalysis method), 1410
__init__() (angr.rust.analyses.rustc_version_identification.RustcVersionI method), 1410
__init__() (angr.rust.analyses.struct_builder.StructBuilder method), 1410
__init__() (angr.rust.analyses.type_db_loader.TypeDBLoader method), 1411
__init__() (angr.rust.knowledge_plugins.clinic_factory.ClinicFactory method), 1412
__init__() (angr.rust.knowledge_plugins.known_structs.KnownStructs method), 1412
__init__() (angr.rust.knowledge_plugins.known_structs.StructMatcher method), 1412
__init__() (angr.rust.knowledge_plugins.librust.Librust method), 1413
__init__() (angr.rust.knowledge_plugins.rust_calling_conventions.RustC method), 1413
__init__() (angr.rust.knowledge_plugins.type_hints.TypeHints method), 1413
__init__() (angr.rust.mixins.cfa_mixin.CFAMixin method), 1417
__init__() (angr.rust.mixins.cfg_transformation_mixin.CFGTransformati method), 1418
__init__() (angr.rust.mixins.dfa_mixin.DFAMixin method), 1419
__init__() (angr.rust.mixins.dfa_mixin.StackDefinition method), 1418
__init__() (angr.rust.mixins.srda_mixin.SRDAMixin method), 1419
__init__() (angr.rust.mixins.ssa_variable_mixin.SSAVariableMixin method), 1420

__init__() (angr.rust.optimization_passes.cleanup_code_remover.CleanupCodeRemover method), 1421
__init__() (angr.rust.optimization_passes.combo_register_rewriter.ComboRegisterRewriter method), 1422
__init__() (angr.rust.optimization_passes.deref_coercion_simplifier.DerefCoercionSimplifier method), 1422
__init__() (angr.rust.optimization_passes.deref_coercion_simplifier.StrCangSimplifierWalker method), 1422
__init__() (angr.rust.optimization_passes.deref_coercion_simplifier.UninlinedDerefCoercionSimplifierUnpeared method), 1422
__init__() (angr.rust.optimization_passes.error_propagation_simplifier.ErrorPropagationSimplifier method), 1423
__init__() (angr.rust.optimization_passes.error_propagation_simplifier.ErrorPropagationWalker method), 1423
__init__() (angr.rust.optimization_passes.function_prototype_inference.FunctionPrototypeInference method), 1423
__init__() (angr.rust.optimization_passes.macro.format_macro_simplifier.FormatMacroSimplifier method), 1424
__init__() (angr.rust.optimization_passes.macro.format_macro_simplifier.FormatWrapperIdentifier method), 1423
__init__() (angr.rust.optimization_passes.macro.format_macro_simplifier.FormatWrapperIdentifier method), 1424
__init__() (angr.rust.optimization_passes.outliners.string_cmp_outliner.StringCmpOutliner method), 1425
__init__() (angr.rust.optimization_passes.outliners.string_literal_outliner.StringLiteralOutliner method), 1425
__init__() (angr.rust.optimization_passes.outliners.string_outliner.StringOutliner method), 1425
__init__() (angr.rust.optimization_passes.outliners.unwrap_outliner.UnwrapOutliner method), 1426
__init__() (angr.rust.optimization_passes.outliners.unwrap_outliner.UnwrapSimplifierState method), 1425
__init__() (angr.rust.optimization_passes.outliners.vec_outliner.VecOutliner method), 1426
__init__() (angr.rust.optimization_passes.pattern_match_simplifier.PatternMatchSimplifier method), 1426
__init__() (angr.rust.optimization_passes.pattern_match_simplifier.PatternMatchWalker method), 1426
__init__() (angr.rust.optimization_passes.pre_pattern_match_simplifier.PrePatternMatchSimplifier method), 1427
__init__() (angr.rust.optimization_passes.redundant_block_remover.RedundantBlockRemover method), 1428
__init__() (angr.rust.optimization_passes.ret_expr_rewriter.RetExprRewriter method), 1428
__init__() (angr.rust.optimization_passes.rust_calling_convention.RustCallingConvention method), 1428
__init__() (angr.rust.optimization_passes.security_check_remover.SecurityCheckRemover method), 1429
__init__() (angr.rust.optimization_passes.str_argument_simplifier.StrArgumentSimplifier method), 1429
__init__() (angr.rust.optimization_passes.struct_instantiation_simplifier.StructBuilder method), 1429
__init__() (angr.rust.optimization_passes.struct_instantiation_simplifier.StructInstantiationSimplifier method), 1430
__init__() (angr.rust.optimization_passes.struct_return_simplifier.StructI method), 1430
__init__() (angr.rust.optimization_passes.utils.CallRewriter method), 1431
__init__() (angr.rust.optimization_passes.utils.SideEffectStatementRewri method), 1430
__init__() (angr.rust.optimization_passes.utils.WalkerEnumVariant method), 1434
__init__() (angr.rust.optimization_passes.utils.RustSimTypeArray method), 1432
__init__() (angr.rust.optimization_passes.utils.RustSimTypeFunction method), 1432
__init__() (angr.rust.optimization_passes.utils.RustSimTypeReference method), 1432
__init__() (angr.rust.optimization_passes.utils.FunctionInferencer_nodes.IfLetNode method), 1436
__init__() (angr.rust.optimization_passes.utils.FunctionInferencer_nodes.PatternMatchNode method), 1436
__init__() (angr.rust.optimization_passes.utils.FormatWrapperIdentifier method), 1437
__init__() (angr.rust.optimization_passes.utils.FormatWrapperIdentifier method), 1438
__init__() (angr.rust.optimization_passes.utils.CallVisitor method), 1448
__init__() (angr.rust.optimization_passes.utils.ErrorRecord method), 1442
__init__() (angr.rust.optimization_passes.utils.SimulationManager method), 1442
__init__() (angr.sim_procedure.SimProcedure method), 1450
__init__() (angr.sim_state.SimState method), 1453
__init__() (angr.sim_state_options.SimStateOptions method), 1458
__init__() (angr.sim_state_options.StateOption method), 1456
__init__() (angr.sim_type.NamedTypeMixin method), 1462
__init__() (angr.sim_type.SimStructValue method), 1472
__init__() (angr.sim_type.SimType method), 1460
__init__() (angr.sim_type.SimTypeArray method), 1467
__init__() (angr.sim_type.SimTypeBool method), 1465
__init__() (angr.sim_type.SimTypeChar method), 1464
__init__() (angr.sim_type.SimTypeFd method), 1465
__init__() (angr.sim_type.SimTypeFunction method), 1469
__init__() (angr.sim_type.SimTypeInt method), 1463
__init__() (angr.sim_type.SimTypeLength method), 1470
__init__() (angr.sim_type.SimTypeNum method), 1463
__init__() (angr.sim_type.SimTypePointer method), 1466
__init__() (angr.sim_type.SimTypeReg method), 1462
__init__() (angr.sim_type.SimTypeString method), 1468

__init__() (*angr.sim_type.SimTypeWideChar method*), 1465
__init__() (*angr.sim_type.SimUnion method*), 1472
__init__() (*angr.sim_type.SimUnionValue method*), 1472
__init__() (*angr.sim_variable.SimVariable method*), 1477
__init__() (*angr.simos.cgc.SimCGC method*), 1486
__init__() (*angr.simos.javavm.SimJavaVM method*), 1486
__init__() (*angr.simos.linux.SimLinux method*), 1488
__init__() (*angr.simos.simos.GlobalDescriptorTable method*), 1490
__init__() (*angr.simos.simos.SimOS method*), 1488
__init__() (*angr.simos.snimmuc_nxp.SimSnimmucNxp method*), 1491
__init__() (*angr.simos.uefi.SimUefi method*), 1491
__init__() (*angr.simos.userland.SimUserland method*), 1491
__init__() (*angr.simos.windows.SimWindows method*), 1492
__init__() (*angr.simos.xbox.SimXbox method*), 1493
__init__() (*angr.slicer.SimLightState method*), 1493
__init__() (*angr.slicer.SimSlicer method*), 1493
__init__() (*angr.state_hierarchy.StateHierarchy method*), 1494
__init__() (*angr.state_plugins.callstack.CallStack method*), 1542
__init__() (*angr.state_plugins.callstack.CallStackAction method*), 1544
__init__() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
__init__() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 1545
__init__() (*angr.state_plugins.edge_hitmap.SimStateEdgeHitmap method*), 1547
__init__() (*angr.state_plugins.filesystem.SimConcreteFilesystem method*), 1550
__init__() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
__init__() (*angr.state_plugins.filesystem.SimHostFilesystem method*), 1550
__init__() (*angr.state_plugins.gdb.GDB method*), 1551
__init__() (*angr.state_plugins.globals.SimStateGlobals method*), 1551
__init__() (*angr.state_plugins.heap.heap_base.SimHeapBase method*), 1556
__init__() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 1556
__init__() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
__init__() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 1559
__init__() (*angr.state_plugins.heap.heap_ptmalloc.PTChunkIterator method*), 1560
__init__() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 1560
__init__() (*angr.state_plugins.history.LambdaAttrIter method*), 1563
__init__() (*angr.state_plugins.history.LambdaIterIter method*), 1563
__init__() (*angr.state_plugins.history.SimStateHistory method*), 1561
__init__() (*angr.state_plugins.history.TreeIter method*), 1563
__init__() (*angr.state_plugins.icicle.IcicleStateTranslationData method*), 1563
__init__() (*angr.state_plugins.icicle.IcicleVMRef method*), 1564
__init__() (*angr.state_plugins.icicle.SimStateIcicle method*), 1564
__init__() (*angr.state_plugins.inspect.BP method*), 1570
__init__() (*angr.state_plugins.inspect.InspectAttrs method*), 1568
__init__() (*angr.state_plugins.inspect.SimInspector method*), 1571
__init__() (*angr.state_plugins.javavm_classloader.SimJavaVmClassload method*), 1573
__init__() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 1573
__init__() (*angr.state_plugins.libc.SimStateLibc method*), 1576
__init__() (*angr.state_plugins.light_registers.SimLightRegisters method*), 1577
__init__() (*angr.state_plugins.log.SimStateLog method*), 1577
__init__() (*angr.state_plugins.loop_data.SimStateLoopData method*), 1577
__init__() (*angr.state_plugins.plugin.SimStatePlugin method*), 1578
__init__() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
__init__() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 1582
__init__() (*angr.state_plugins.scratch.SimStateScratch method*), 1583
__init__() (*angr.state_plugins.sim_action.SimAction method*), 1584
__init__() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1586
__init__() (*angr.state_plugins.sim_event.SimEvent method*), 1587
__init__() (*angr.state_plugins.solver.SimSolver method*), 1588
__init__() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 1596

__init__() (angr.state_plugins.trace_additions.ChallRespInfo method), 1597
__init__() (angr.state_plugins.trace_additions.FormatInfoDimiConstraint method), 1597
__init__() (angr.state_plugins.trace_additions.FormatInfoInfoStr method), 1596
__init__() (angr.state_plugins.trace_additions.FormatInfoStrToInt method), 1596
__init__() (angr.state_plugins.trace_additions.ZenPlugin method), 1598
__init__() (angr.state_plugins.uc_manager.SimUCManager method), 1598
__init__() (angr.state_plugins.unicorn_engine.AggressiveConcretization method), 1603
__init__() (angr.state_plugins.unicorn_engine.Unicorn method), 1603
__init__() (angr.state_plugins.unicorn_engine.Uniwrapper method), 1603
__init__() (angr.state_plugins.view.SimMemView method), 1605
__init__() (angr.state_plugins.view.StructMode method), 1607
__init__() (angr.storage.file.SimFile method), 1611
__init__() (angr.storage.file.SimFileBase method), 1610
__init__() (angr.storage.file.SimFileDescriptor method), 1615
__init__() (angr.storage.file.SimFileDescriptorDuplex method), 1616
__init__() (angr.storage.file.SimFileStream method), 1612
__init__() (angr.storage.file.SimPackets method), 1613
__init__() (angr.storage.file.SimPacketsSlots method), 1617
__init__() (angr.storage.file.SimPacketsStream method), 1613
__init__() (angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method), 1633
__init__() (angr.storage.memory_mixins.convenient_mappings_mixin.ConvenientMappingsMixin method), 1634
__init__() (angr.storage.memory_mixins.default_filler_mixin.ExplicitFillerMixin method), 1635
__init__() (angr.storage.memory_mixins.default_filler_mixin.SpecialFillerMixin method), 1635
__init__() (angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method), 1636
__init__() (angr.storage.memory_mixins.keyvalue_memory_mixin.KeyValueMemoryMixin method), 1638
__init__() (angr.storage.memory_mixins.keyvalue_memory_mixin.TypedVariable method), 1637
__init__() (angr.storage.memory_mixins.label_merger_mixin.LabelMergerMixin method), 1638
__init__() (angr.storage.memory_mixins.memory_mixin.MemoryMixin method), 1638
__init__() (angr.storage.memory_mixins.multi_value_merger_mixin.Mul method), 1641
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_m method), 1641
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_m method), 1641
__init__() (angr.storage.memory_mixins.paged_memory.page_backer_m method), 1641
__init__() (angr.storage.memory_mixins.paged_memory.paged_memory_ method), 1643
__init__() (angr.storage.memory_mixins.paged_memory.paged_memory_ method), 1642
__init__() (angr.storage.memory_mixins.paged_memory.pages.history_tr method), 1648
__init__() (angr.storage.memory_mixins.paged_memory.pages.list_page. method), 1649
__init__() (angr.storage.memory_mixins.paged_memory.pages.multi_val method), 1649
__init__() (angr.storage.memory_mixins.paged_memory.pages.mv_list_p method), 1651
__init__() (angr.storage.memory_mixins.paged_memory.pages.permissio method), 1651
__init__() (angr.storage.memory_mixins.paged_memory.pages.refcount_ method), 1652
__init__() (angr.storage.memory_mixins.paged_memory.pages.symbolic_ method), 1652
__init__() (angr.storage.memory_mixins.paged_memory.pages.ultra_pag method), 1654
__init__() (angr.storage.memory_mixins.paged_memory.stack_allocation method), 1655
__init__() (angr.storage.memory_mixins.regioned_memory.abstract_add method), 1659
__init__() (angr.storage.memory_mixins.regioned_memory.region_data. method), 1659
__init__() (angr.storage.memory_mixins.regioned_memory.region_data. method), 1660
__init__() (angr.storage.memory_mixins.regioned_memory.region_data. method), 1660
__init__() (angr.storage.memory_mixins.regioned_memory.region_meta_ method), 1662
__init__() (angr.storage.memory_mixins.regioned_memory.region_meta_ method), 1662
__init__() (angr.storage.memory_mixins.regioned_memory.region_meta_ method), 1661
__init__() (angr.storage.memory_mixins.regioned_memory.regioned_add method), 1662
__init__() (angr.storage.memory_mixins.regioned_memory.regioned_me method), 1663
__init__() (angr.storage.memory_mixins.size_resolution_mixin.SizeConc method), 1665
__init__() (angr.storage.memory_mixins.slotted_memory.SlottedMemory method), 1665
__init__() (angr.storage.memory_mixins.top_merger_mixin.TopMergerM method), 1666

__init__() (angr.storage.memory_mixins.underconstrained_init_UC_constraints_MixIn method), 1666
__init__() (angr.storage.memory_object.SimLabeledMemoryObject method), 1667
__init__() (angr.storage.memory_object.SimMemoryObject method), 1666
__init__() (angr.tablespecs.StringTableSpec method), 1668
__init__() (angr.utils.ail.HasExprWalker method), 1670
__init__() (angr.utils.ail_serialization.BlockPool method), 1671
__init__() (angr.utils.balancer.Balancer method), 1673
__init__() (angr.utils.doms.IncrementalDominators method), 1677
__init__() (angr.utils.dynamic_dictlist.DynamicDictList method), 1678
__init__() (angr.utils.graph.ContainerNode method), 1684
__init__() (angr.utils.graph.DirectedGraphHelper method), 1688
__init__() (angr.utils.graph.Dominators method), 1685
__init__() (angr.utils.graph.LinkedListNode method), 1687
__init__() (angr.utils.graph.PostDominators method), 1685
__init__() (angr.utils.graph.SCCPlaceholder method), 1685
__init__() (angr.utils.graph.TemporaryNode method), 1684
__init__() (angr.utils.ins_addr_list.InsAddrList method), 1689
__init__() (angr.utils.mp.Initializer method), 1694
__init__() (angr.utils.orderedset.OrderedSet method), 1695
__init__() (angr.utils.smart_cache.SmartLRUCache method), 1695
__init__() (angr.utils.ssa.AILWhitelistExprTypeWalker method), 1695
__init__() (angr.utils.ssa.combined_uses_collector.VVarAndTmpUsesCollector method), 1698
__init__() (angr.utils.ssa.tmp_uses_collector.TmpUsesCollector method), 1699
__init__() (angr.utils.ssa.vvar_extra_defs_collector.FindExtraDefs method), 1699
__init__() (angr.utils.ssa.vvar_uses_collector.VVarUsesCollector method), 1699
__init__() (angr.utils.tagged_interval_map.TaggedIntervalMap method), 1700
__init__() (angr.vaults.Vault method), 1704
__init__() (angr.vaults.VaultDict method), 1704
__init__() (angr.vaults.VaultDir method), 1705
__init__() (angr.vaults.VaultDirShelf method), 1705
__init__() (angr.vaults.VaultPickler method), 1704
__init__() (angr.vaults.VaultShelf method), 1705
__new__() (angr.ailment.Block class method), 210
__new__() (angr.ailment.Manager class method), 212
__new__() (angr.ailment.block.Block class method), 214
__new__() (angr.ailment.expression.Array static method), 218
__new__() (angr.ailment.expression.BasePointerOffset static method), 218
__new__() (angr.ailment.expression.BinaryOp static method), 218
__new__() (angr.ailment.expression.Call static method), 218
__new__() (angr.ailment.expression.ComboRegister static method), 219
__new__() (angr.ailment.expression.Const static method), 219
__new__() (angr.ailment.expression.Convert static method), 219
__new__() (angr.ailment.expression.DirtyExpression static method), 219
__new__() (angr.ailment.expression.Extract static method), 220
__new__() (angr.ailment.expression.FunctionLikeMacro static method), 220
__new__() (angr.ailment.expression.ITE static method), 218
__new__() (angr.ailment.expression.Insert static method), 220
__new__() (angr.ailment.expression.Let static method), 220
__new__() (angr.ailment.expression.Load static method), 221
__new__() (angr.ailment.expression.Macro static method), 221
__new__() (angr.ailment.expression.MultiStatementExpression static method), 221
__new__() (angr.ailment.expression.Phi static method), 221
__new__() (angr.ailment.expression.Register static method), 221
__new__() (angr.ailment.expression.Reinterpret static method), 221
__new__() (angr.ailment.expression.RustEnum static method), 222
__new__() (angr.ailment.expression.StackBaseOffset static method), 222
__new__() (angr.ailment.expression.StringLiteral static method), 222
__new__() (angr.ailment.expression.Struct static method), 222
__new__() (angr.ailment.expression.Tmp static method), 222

__new__() (*angr.ailment.expression.UnaryOp static method*), 222
__new__() (*angr.ailment.expression.VEXCCallExpression static method*), 223
__new__() (*angr.ailment.expression.VirtualVariable static method*), 223
__new__() (*angr.ailment.manager.Manager class method*), 224
__new__() (*angr.ailment.statement.Assignment static method*), 224
__new__() (*angr.ailment.statement.CAS static method*), 224
__new__() (*angr.ailment.statement.ConditionalJump static method*), 224
__new__() (*angr.ailment.statement.DirtyStatement static method*), 225
__new__() (*angr.ailment.statement.Jump static method*), 225
__new__() (*angr.ailment.statement.Label static method*), 225
__new__() (*angr.ailment.statement.NoOp static method*), 225
__new__() (*angr.ailment.statement.Return static method*), 225
__new__() (*angr.ailment.statement.SideEffectStatement static method*), 225
__new__() (*angr.ailment.statement.Store static method*), 226
__new__() (*angr.ailment.statement.WeakAssignment static method*), 226
__new__() (*angr.rustylib.Segment class method*), 1439
__new__() (*angr.rustylib.SegmentList class method*), 1439

## A

AArch64MachOGotResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.aarch64_macho_got*), 331
abort (*class in angr.procedures.libc.abort*), 1353
abort() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 764
absolutize() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionMap method*), 1661
absorb() (*angr.sim_manager.SimulationManager method*), 1445
absorb_successor_into() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 629
abstract() (*angr.analyses.typehoon.simple_solver.SimpleSolver static method*), 904
AbstractAddressDescriptor (*class in angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor*), 1659
AbstractLocation (*class in angr.storage.memory_mixins.regioned_memory.region_meta_mix*), 1661
AbstractMemory (*class in angr.storage.memory_mixins*), 1617
AbstractMergerMixin (*class in angr.storage.memory_mixins.regioned_memory.abstract_merger_*), 1659
accept (*class in angr.procedures.posix.accept*), 1370
access (*class in angr.procedures.libc.access*), 1353
access (*class in angr.procedures.linux_kernel.access*), 1364
accessed_data_references (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
AccessingZeroPageError, 1603
acquire() (*angr.misc.picklable_lock.PicklableLock method*), 1317
acquire_shared() (*angr.storage.memory_mixins.paged_memory.pages.re method*), 1652
acquire_unique() (*angr.storage.memory_mixins.paged_memory.pages.re method*), 1652
act_instruction() (*angr.analyses.smc.TraceClassifier method*), 892
act_mem_write() (*angr.analyses.smc.TraceClassifier method*), 892
Action (*in module angr.state_plugins.inspect*), 1570
action() (*angr.state_plugins.inspect.SimInspector method*), 1571
actions (*angr.state_plugins.history.SimStateHistory property*), 1562
actions (*angr.state_plugins.log.SimStateLog property*), 1577
actions_of_type() (*angr.state_plugins.log.SimStateLog method*), 1577
ActionsMixinHigh (*class in angr.storage.memory_mixins.actions_mixin*), 1623
ActionsMixinLow (*class in angr.storage.memory_mixins.actions_mixin*), 1623
activate() (*angr.misc.plugins.PluginPreset method*), 1310
active_workers (*angr.distributed.server.Server property*), 994
Add (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransfor attribute*), 334
Add (*angr.engines.light.ArithmeticExpression attribute*), 1005
Add (*angr.engines.light.data.ArithmeticExpression attribute*), 1009
Add (*class in angr.analyses.typehoon.typevars*), 915
add() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 1104
add() (*angr.procedures.definitions.SimLibrary method*),

1326
add() (angr.procedures.definitions.SimTypeCollection method), 1324
add() (angr.sim_state_options.SimStateOptions method), 1458
add() (angr.state_plugins.solver.SimSolver method), 1591
add() (angr.utils.ail_serialization.BlockPool method), 1671
add() (angr.utils.tagged_interval_map.TaggedIntervalMap method), 1700
add_action() (angr.state_plugins.history.SimStateHistory method), 1562
add_action() (angr.state_plugins.log.SimStateLog method), 1577
add_alias() (angr.procedures.definitions.SimLibrary method), 1327
add_alias() (angr.procedures.definitions.SimSyscallLibrary method), 1331
add_all_from_dict() (angr.procedures.definitions.SimLibrary method), 1326
add_block_to_whitelist() (angr.annocfg.AnnotatedCFG method), 957
add_bookmark() (angr.knowledge_plugins.bookmarks.Bookmarks method), 1149
add_breakpoint() (angr.emulator.Emulator method), 995
add_breakpoint() (angr.state_plugins.inspect.SimInspect method), 1572
add_call_stack_arg_offset() (angr.analyses.decompiler.stackarg_offset_manager.StackArgOffsetManager method), 660
add_cfuncall_config() (angr.analyses.decompiler.structured_codegen.c_serialize.SerializedContext method), 689
add_constraint() (angr.analyses.typehoon.simple_solver.Sketch method), 901
add_constraints() (angr.sim_state.SimState method), 1454
add_data_seg() (angr.analyses.cfg.cfg_fast.DecodingAssumption method), 315
add_def() (angr.analyses.ddg.LiveDefinitions method), 362
add_def() (angr.knowledge_plugins.key_definitions.liveness.Liveness method), 1257
add_def() (angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method), 1259
add_default_plugin() (angr.misc.plugins.PluginPreset method), 1319
add_default_value_generator() (angr.engines.soot.values.arrayref.SimSootValue_ArrayBaseRef method), 1059
add_defs() (angr.analyses.ddg.LiveDefinitions method), 362
add_dependencies_for_concrete_pointers_of() (angr.analyses.reaching_definitions.dep_graph.DepGraph method), 836
add_edge() (angr.analyses.decompiler.region_overlay.RegionOverlay method), 628
add_edge() (angr.analyses.reaching_definitions.dep_graph.DepGraph method), 835
add_edge() (angr.analyses.typehoon.simple_solver.Sketch method), 901
add_edge() (angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method), 1183
add_edge_by_key() (angr.knowledge_plugins.cfg.spilling_cfg.SpillingCF method), 1184
add_edge_to_buffer() (in module angr.utils.formatting), 1679
add_edges_to_condition() (angr.analyses.decompiler.optimization_passes.duplication_rever method), 569
add_event() (angr.state_plugins.history.SimStateHistory method), 1562
add_event() (angr.state_plugins.log.SimStateLog method), 1577
add_exit_to_whitelist() (angr.annocfg.AnnotatedCFG method), 957
add_final_job() (angr.analyses.vfg.CallAnalysis method), 934
add_function() (angr.analyses.cfg.cfb.CFBlanket method), 306
add_function_edge() (angr.analyses.cfg.cfg_fast.CFGJob method), 320
add_goal() (angr.exploration_techniques.director.Director method), 1101
add_heap_use() (angr.analyses.reaching_definitions.rd_state.ReachingDe method), 865
add_heap_use() (angr.knowledge_plugins.key_definitions.live_definitions. method), 1255
add_heap_use_by_def() (angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi method), 1256
add_heap_use_by_defs() (angr.analyses.reaching_definitions.rd_state.ReachingDefinitions method), 865
add_history() (angr.state_hierarchy.StateHierarchy method), 1484
add_job() (angr.analyses.cfg.cfg_fast.PendingJobs method), 316
add_job() (angr.analyses.forward_analysis.job_info.JobInfo method), 764
add_jumpout_site() (angr.knowledge_plugins.functions.function.Functio method), 1214
add_jumptable() (angr.knowledge_plugins.cfg.indirect_jump.IndirectJum

*method*), 1177
add_key_func_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1226
add_label() (*angr.analyses.reassembler.Reassembler method*), 876
add_label() (*in module angr.analyses.typehoon.typevars*), 919
add_labels() (*in module angr.analyses.decompiler.utils*), 723
add_loop() (*angr.annocfg.AnnotatedCFG method*), 957
add_mapping() (*angr.analyses.decompiler.structured_codegen.base.InstructionMapping method*), 669
add_mapping() (*angr.analyses.decompiler.structured_codegen.base.StructuredCodeMapping method*), 668
add_memory_data() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1171
add_memory_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1256
add_memory_use_by_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 865
add_memory_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1256
add_memory_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 865
add_node() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 627
add_node() (*angr.analyses.decompiler.structurer_nodes.SequenceNode method*), 711
add_node() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 835
add_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1167
add_node() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1182
add_node_successor() (*angr.utils.graph.DirectedGraphHelper method*), 1688
add_nonreturning_function() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 316
add_null() (*angr.tablespecs.StringTableSpec method*), 1668
add_number_mapping() (*angr.procedures.definitions.SimSyscallLibrary method*), 1330
add_number_mapping_from_dict() (*angr.procedures.definitions.SimSyscallLibrary method*), 1330
add_obj() (*angr.analyses.cfg.cfb.CFBlanket method*), 306
add_object() (*angr.keyed_region.KeyedRegion method*), 1117
add_object() (*angr.keyed_region.RegionObject method*), 1116
add_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
add_patch_obj() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
add_pointer() (*angr.tablespecs.StringTableSpec method*), 1668
add_regioned_address() (*angr.knowledge_plugins.memory_mixins.regioned_memory.abstract_address.RegionedAddress method*), 1659
add_register_mapping() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 864
add_register_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1255
add_register_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1255
add_register_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 864
add_replacement() (*angr.knowledge_plugins.propagations.states.PropagationState method*), 1274
add_retout_site() (*angr.knowledge_plugins.functions.function.Function method*), 1214
add_returning_function() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 316
add_simple() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
add_stack_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 864
add_stack_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1255
add_stack_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1255
add_stack_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 865
add_state() (*angr.state_hierarchy.StateHierarchy method*), 1494
add_statements_to_whitelist() (*angr.annocfg.AnnotatedCFG method*), 957
add_string() (*angr.analyses.decompiler.notes.deobfuscated_strings.DeobfuscatedStrings method*), 539
add_string() (*angr.tablespecs.StringTableSpec method*), 1668
add_successor() (*angr.engines.successors.SimSuccessors method*), 1064
add_tmp_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 864

add_tmp_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1256
add_tmp_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1256
add_tmp_use_by_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 864
add_transitions() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink method*), 349
add_type_constraint() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 929
add_type_constraint_for_function() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 929
add_type_hint() (*angr.rust.knowledge_plugins.type_hints.TypeHints method*), 1413
add_type_variable() (*angr.analyses.typehoon.typevars.TypeVariables method*), 918
add_use() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 863
add_use() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1253
add_use() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1262
add_use_by_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 864
add_use_by_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1253
add_value() (*angr.storage.memory_mixins.paged_memory.page_backer_mixins.MultiValuedMemory method*), 1649
add_variable() (*angr.keyed_region.KeyedRegion method*), 1117
add_variable() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 1193
add_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1283
add_variable_list() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 1193
add_vvar_def() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 886
add_vvar_use() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 885
add_xref() (*angr.analyses.xrefs.SimEngineXRefsVEX method*), 937
add_xref() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
add_xrefs() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
added_constraints (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
added_constraints (*angr.state_plugins.InspectAttrs attribute*), 1498
AddN (*class in angr.analyses.typehoon.typevars*), 919
addr (*angr.analyses.CFGFast.LiveDefinitions attribute*), 210
addr (*angr.ailment.block.Block attribute*), 214
addr (*angr.analyses.decompiler.region_overlay.RegionBound property*), 623
addr (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 626
addr (*angr.analyses.disassembly.DisassemblyPiece attribute*), 739
addr (*angr.analyses.disassembly.OperandPiece attribute*), 742
addr (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
addr (*angr.sim_state.SimState property*), 1454
addr (*angr.state_plugins.history.SimStateHistory property*), 1561
addr_invalid() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
addr_to_instruction_addr() (*angr.knowledge_plugins.functions.function.Function method*), 1216
addr_type (*angr.analyses.cfg.cfg_base.CFGBase attribute*), 307
addr_type (*angr.analyses.cfg.cfg_emulated.CFGEmulated attribute*), 311
addr_type (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 321
addr_type (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot attribute*), 325
addr_type (*angr.analyses.cfg.CFGBase attribute*), 292
addr_type (*angr.analyses.cfg.CFGEmulated attribute*), 295
addr_type (*angr.analyses.cfg.CFGFast attribute*), 299
addr_type (*angr.analyses.cfg.CFGFastSoot attribute*), 303
addr_type (*angr.analyses.CFGEmulated attribute*), 238
addr_type (*angr.analyses.CFGFast attribute*), 242
addr_type (*angr.analyses.CFGFastSoot attribute*), 246
addr_type (*angr.knowledge_plugins.cfg.cfg_model.CFGModel property*), 1166
addr_type (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1182
addr_type (*angr.knowledge_plugins.cfg.spilling_digraph.SpillingDiGraph property*), 1188
address (*angr.analyses.reaching_definitions.function_handler.FunctionCallMessage attribute*), 840
address (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
address (*angr.block.CapstoneInsn property*), 960
address (*angr.block.DisassemblerInsn property*), 959
address (*angr.block.PCodeInsn property*), 960
address (*angr.engines.pcode.lifter.PcodeDisassemblerInsn

*property*), 1037
address (*angr.knowledge_plugins.cfg.memory_data.MemoryData property*), 1178
address (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
address (*angr.state_plugins.InspectAttrs attribute*), 1498
ADDRESS_CONCRETIZATION (*angr.state_plugins.inspect.EventType attribute*), 1565
address_concretization_action (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
address_concretization_action (*angr.state_plugins.InspectAttrs attribute*), 1499
address_concretization_add_constraints (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
address_concretization_add_constraints (*angr.state_plugins.InspectAttrs attribute*), 1499
address_concretization_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
address_concretization_expr (*angr.state_plugins.InspectAttrs attribute*), 1499
address_concretization_memory (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
address_concretization_memory (*angr.state_plugins.InspectAttrs attribute*), 1499
address_concretization_result (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
address_concretization_result (*angr.state_plugins.InspectAttrs attribute*), 1499
address_concretization_strategy (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
address_concretization_strategy (*angr.state_plugins.InspectAttrs attribute*), 1499
AddressConcretizationMixin (*class in angr.storage.memory_mixins.address_concretization_mixin*), 1633
AddressOperand (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 334
AddressPair (*class in angr.protos.clinic_pb2*), 1396
AddressTransformation (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 334
AddressTransformationTypes (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 334
AddressTuple (*class in angr.protos.clinic_pb2*), 1396
AddressWrapper (*class in angr.storage.memory_mixins.regioned_memory.region_data*), 1659
addrs_for_hash() (*angr.storage.memory_mixins.convenient_mappings_m method*), 1634
addrs_for_name() (*angr.storage.memory_mixins.convenient_mappings_m method*), 1634
AddrToLabelEntry (*class in angr.protos.codegen_pb2*), 1397
ADDS_EXITS (*angr.procedures.posix.pthread.pthread_create attribute*), 1373
ADDS_EXITS (*angr.sim_procedure.SimProcedure attribute*), 1450
ADDS_EXITS (*angr.SimProcedure attribute*), 190
ADivConstAddAMulNDivConst (*class in angr.analyses.decompiler.peephole_optimizations.a_div_const_ad 600*
adjust_mapping_positions() (*angr.analyses.decompiler.structured_codegen.base.BaseStructur static method*), 669
AFTER (*angr.state_plugins.inspect.When attribute*), 1570
AFTER_AIL_GRAPH_CREATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 586
AFTER_AIL_GRAPH_CREATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 552
after_call() (*angr.procedures.linux_loader.sim_loader.IFuncResolver method*), 1368
after_call() (*angr.procedures.stubs.caller.Caller method*), 1378
AFTER_GLOBAL_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 587
AFTER_GLOBAL_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 552
AFTER_MAKING_CALLSITES (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 587
AFTER_MAKING_CALLSITES (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 552
AFTER_SINGLE_BLOCK_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 587
AFTER_SINGLE_BLOCK_SIMPLIFICATION (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 552

AFTER_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 587
AFTER_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 552
AFTER_STRUCTURING (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 587
AFTER_STRUCTURING (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 552
AFTER_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 587
AFTER_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 552
AgentKind (*class in angr.llm_client*), 1293
AggressiveConcretizationAnnotation (*class in angr.state_plugins.unicorn_engine*), 1603
ail_block_from_stmts() (*in module angr.analyses.decompiler.optimization_passes.duplication_reverter.utils*), 573
ail_block_idx (*angr.state_plugins.scratch.SimStateScratch property*), 1583
ail_call_state() (*in module angr.engines.ail.setup*), 1004
ail_const_to_be() (*in module angr.utils.endness*), 1678
AIL_GRAPH_CONVERSION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
ail_graph_similarity() (*in module angr.analyses.decompiler.optimization_passes.duplication_reverter.similarity*), 572
ail_similarity_to_orig_blocks() (*in module angr.analyses.decompiler.optimization_passes.duplication_reverter.similarity*), 572
AILBlockCallCounter (*class in angr.analyses.decompiler.counters.call_counter*), 411
AILBlockRewriter (*class in angr.ailment.block_walker*), 216
AILBlockSplit (*class in angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph*), 568
AILBlockTempCollector (*class in angr.analyses.decompiler.ail_simplifier*), 397
AILBlockViewer (*class in angr.ailment.block_walker*), 216
AILBlockWalker (*class in angr.ailment.block_walker*), 215
AILCallable (*class in angr.ail_callable*), 208
AILCallCounter (*class in angr.analyses.decompiler.counters.call_counter*), 411
AILCallStack (*class in angr.engines.ail.callstack*), 1003
AILCodeLocation (*class in angr.code_location*), 986
AilEdgeClassification (*class in angr.protos.ail_types_pb2*), 1393
AilEdgeData (*class in angr.protos.ail_types_pb2*), 1393
AILExprClassifier (*class in angr.analyses.decompiler.condition_processor*), 408
AilGraph (*class in angr.protos.ail_types_pb2*), 1393
AILGraphWalker (*class in angr.analyses.decompiler.graph_walker*), 398
AILMergeGraph (*class in angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph*), 568
AILMixin (*class in angr.engines.ail.engine_successors*), 1004
AILPurityAnalysis (*class in angr.analyses.purity.analysis*), 801
AILSimplifier (*class in angr.analyses.decompiler.ail_simplifier*), 397
AILWhitelistExprTypeWalker (*class in angr.utils.ssa*), 1695
align (*angr.sim_type.SimStruct property*), 1471
ALIGNMENT (*angr.analyses.codecave.CodeCaveClassification attribute*), 352
Alignment (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
Alignment (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
Alignment (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
alignment (*angr.rust.sim_type.RustSimTypeInt property*), 1431
alignment (*angr.sim_type.SimStruct property*), 1471
alignment (*angr.sim_type.SimType property*), 1460
alignment (*angr.sim_type.SimTypeArray property*), 1468
alignment (*angr.sim_type.SimTypeBitfield property*), 1474
alignment (*angr.sim_type.SimTypeDouble property*), 1471
alignment (*angr.sim_type.SimTypeEnum property*), 1473
alignment (*angr.sim_type.SimTypeString property*), 1468
alignment (*angr.sim_type.SimTypeWString property*), 1469
alignment (*angr.sim_type.SimUnion property*), 1472
alignment (*angr.sim_type.TypeRef property*), 1461
ALL (*angr.sim_manager.SimulationManager attribute*), 1441

ALL (*angr.SimulationManager attribute*), 199
all_bytes_changed_in_history() (*angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin.HistoryTrackingMixin method*), 1648
all_constants (*angr.engines.pcode.lifter.IRSB property*), 1040
all_definitions (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 868
all_definitions (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAM property*), 886
all_exprs() (*angr.knowledge_plugins.propagations.prop_value.PropValue method*), 1269
all_flavors() (*angr.knowledge_plugins.structured_code.StructuredCodeManager method*), 1279
all_jobs() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 316
all_objects (*angr.state_plugins.sim_action.SimAction property*), 1584
all_objects (*angr.state_plugins.sim_action.SimActionConstraint property*), 1585
all_objects (*angr.state_plugins.sim_action.SimActionData property*), 1585
all_objects (*angr.state_plugins.sim_action.SimActionExit property*), 1585
all_objects (*angr.state_plugins.sim_action.SimActionOperation property*), 1585
all_of() (*in module angr.analyses.decompiler.known_patterns.gating*), 515
ALL_PATTERNS (*in module angr.analyses.decompiler.known_patterns.registry*), 524
all_set() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap.SymbolicBitmap method*), 1654
all_stashes (*angr.analyses.Veritesting attribute*), 272
all_stashes (*angr.analyses.veritesting.Veritesting attribute*), 931
all_successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 768
all_successors() (*angr.state_hierarchy.StateHierarchy method*), 1494
ALL_TABLES (*angr.angrdb.AngrDB attribute*), 938
ALL_TABLES (*angr.angrdb.db.AngrDB attribute*), 940
all_uses (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 868
alloc() (*angr.calling_conventions.AllocHelper method*), 963
allocate (*class in angr.procedures.cgc.allocate*), 1323
allocate() (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator method*), 857
allocate() (*angr.knowledge_plugins.custom_strings.CustomStrings method*), 1191
allocate() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 1556
allocate_memory() (*angr.procedures.posix.mmap.mmap method*), 1373
allocate_memory() (*angr.procedures.win32.VirtualAlloc.VirtualAlloc method*), 1383
allocate_stack_pages() (*angr.storage.memory_mixins.paged_memory.stack_allocation_m method*), 1655
allocated_addresses (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocato property*), 857
allocated_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFre method*), 1558
AllocHelper (*class in angr.calling_conventions*), 962
AllocObjectManager (*class in angr.procedures.java_jni.object_operations*), 1347
AllOf (*class in angr.analyses.decompiler.known_patterns.gating*), 513
allow_gaps (*angr.analyses.decompiler.known_patterns.dsl.PStmtSeq attribute*), 505
allow_gaps (*angr.analyses.decompiler.known_patterns.PStmtSeq attribute*), 475
ALT_NAMES (*angr.procedures.libstdcpp.std____throw_bad_alloc.std____thr attribute*), 1363
ALT_NAMES (*angr.procedures.libstdcpp.std____throw_bad_cast.std____thro attribute*), 1363
ALT_NAMES (*angr.procedures.libstdcpp.std____throw_length_error.std____ attribute*), 1363
ALT_NAMES (*angr.procedures.libstdcpp.std____throw_logic_error.std____th attribute*), 1363
ALT_NAMES (*angr.procedures.libstdcpp.std__terminate.std__terminate attribute*), 1363
ALT_NAMES (*angr.SimProcedure attribute*), 1450
ALT_NAMES (*angr.SimProcedure attribute*), 190
ALWAYS (*angr.analyses.decompiler.structuring.phoenix.MultiStmtExprMode attribute*), 717
AlwaysUpdateVisitor (*angr.analyses.calling_convention.calling_convention.Upda attribute*), 285
ambiguous (*angr.analyses.decompiler.edits.resolve.ResolvedVariable attribute*), 447
ambiguous (*angr.analyses.decompiler.edits.ResolvedVariable attribute*), 428
AmbiguousFunctionError, 437
AMD64CCallRewriter (*class in angr.analyses.decompiler.ccall_rewriters.amd64_ccalls*), 402
AMD64DirtyRewriter (*class in angr.analyses.decompiler.dirty_rewriters.amd64_dirty*), 421
AMD64ElfGotResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got*), 331
amd64g_calculate_condition() (*in module angr.engines.vex.claripy.ccall*), 1072

amd64g_calculate_mmx_pmaddwd() (*in module angr.engines.vex.claripy.ccall*), 1072
amd64g_calculate_RCL() (*in module angr.engines.vex.claripy.ccall*), 1072
amd64g_calculate_RCR() (*in module angr.engines.vex.claripy.ccall*), 1072
amd64g_calculate_rflags_all() (*in module angr.engines.vex.claripy.ccall*), 1072
amd64g_calculate_rflags_c() (*in module angr.engines.vex.claripy.ccall*), 1072
amd64g_check_fldcw() (*in module angr.engines.vex.claripy.ccall*), 1071
amd64g_check_ldmxcsr() (*in module angr.engines.vex.claripy.ccall*), 1071
amd64g_check_ldmxcsr() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_create_fpucw() (*in module angr.engines.vex.claripy.ccall*), 1072
amd64g_create_mxcsr() (*in module angr.engines.vex.claripy.ccall*), 1071
amd64g_create_mxcsr() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_CPUID_avx2() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_CPUID_avx512_KNL() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_CPUID_avx512_SKX() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_CPUID_avx_and_cx16() (*in module angr.engines.vex.heavy.dirty*), 1076
amd64g_dirtyhelper_CPUID_baseline() (*in module angr.engines.vex.heavy.dirty*), 1076
amd64g_dirtyhelper_FINIT() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_IN() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_OUT() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_RDMSR() (*in module angr.engines.vex.heavy.dirty*), 1078
amd64g_dirtyhelper_RDTSC() (*in module angr.engines.vex.heavy.dirty*), 1076
amd64g_dirtyhelper_SxDT() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_XGETBV() (*in module angr.engines.vex.heavy.dirty*), 1078
amd64g_dirtyhelper_XRSTOR_COMPONENT_1_EXCLUDING_XMMREGS() (*in module angr.engines.vex.heavy.dirty*), 1077
amd64g_dirtyhelper_XSAVE_COMPONENT_1_EXCLUDING_XMMREGS() (*in module angr.engines.vex.heavy.dirty*), 1077
AMD64PeIatResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.amd64_pe_iat*), 332
AMulConstDivShrConst (*class in angr.analyses.decompiler.peephole_optimizations.a_mul_const_d*), 600
AMulConstSubA (*class in angr.analyses.decompiler.peephole_optimizations.a_mul_const_s*), 600
analyses (*angr.project.Project property*), 1390
AnalysesHub (*class in angr.analyses.analysis*), 275
AnalysesHubWithDefault (*class in angr.analyses.analysis*), 276
Analysis (*class in angr.analyses.analysis*), 276
AnalysisFactory (*class in angr.analyses.analysis*), 276
AnalysisLogEntry (*class in angr.analyses.analysis*), 275
AnalysisTask (*class in angr.analyses.vfg*), 933
analyze() (*angr.analyses.callee_cleanup_finder.CalleeCleanupFinder method*), 283
analyze() (*angr.analyses.code_tagging.CodeTagging method*), 351
analyze() (*angr.analyses.decompiler.optimization_passes.optimization_pa method*), 587
analyze() (*angr.analyses.decompiler.optimization_passes.optimization_pa method*), 588
analyze() (*angr.analyses.decompiler.optimization_passes.optimization_pa method*), 589
analyze() (*angr.analyses.decompiler.semantic_naming.array_index_nami method*), 647
analyze() (*angr.analyses.decompiler.semantic_naming.boolean_naming.B method*), 647
analyze() (*angr.analyses.decompiler.semantic_naming.call_result_namin method*), 648
analyze() (*angr.analyses.decompiler.semantic_naming.naming_base.Sem method*), 648
analyze() (*angr.analyses.decompiler.semantic_naming.orchestrator.Sema method*), 650
analyze() (*angr.analyses.decompiler.semantic_naming.pointer_naming.Po method*), 651
analyze() (*angr.analyses.decompiler.semantic_naming.region_loop_coun method*), 651
analyze() (*angr.analyses.decompiler.semantic_naming.size_naming.SizeN method*), 652
analyze() (*angr.analyses.deobfuscator.api_obf_finder.APIObfuscationFin method*), 735
analyze() (*angr.analyses.deobfuscator.api_obf_type2_finder.APIObfuscat method*), 736
analyze() (*angr.analyses.deobfuscator.string_obf_finder.StringObfuscatio method*), 738
analyze() (*angr.analyses.pathfinder.Pathfinder method*), 795
analyze() (*angr.analyses.unpacker.obfuscation_detector.ObfuscationDete method*), 922
analyze() (*angr.analyses.unpacker.packing_detector.PackingDetector method*), 922
analyze() (*angr.analyses.vtable.VtableFinder method*), 937

analyze_transmit() (*angr.state_plugins.trace_additions.ZenPlugin method*), 1598
ancestors() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 626
And (*angr.engines.light.ArithmeticExpression attribute*), 1006
And (*angr.engines.light.data.ArithmeticExpression attribute*), 1009
angr
    module, 155
angr.ail_callable
    module, 208
angr.ailment
    module, 208
angr.ailment.block
    module, 214
angr.ailment.block_walker
    module, 215
angr.ailment.constant
    module, 216
angr.ailment.converter_common
    module, 216
angr.ailment.converter_pcode
    module, 217
angr.ailment.converter_vex
    module, 217
angr.ailment.expression
    module, 217
angr.ailment.manager
    module, 224
angr.ailment.statement
    module, 224
angr.ailment.tagged_object
    module, 226
angr.ailment.utils
    module, 227
angr.analyses
    module, 228
angr.analyses.analysis
    module, 275
angr.analyses.backward_slice
    module, 277
angr.analyses.binary_optimizer
    module, 279
angr.analyses.bindiff
    module, 280
angr.analyses.boyscout
    module, 283
angr.analyses.callee_cleanup_finder
    module, 283
angr.analyses.calling_convention
    module, 283
angr.analyses.calling_convention.calling_convention
    module, 285
angr.analyses.calling_convention.fact_collector
    module, 286
angr.analyses.calling_convention.utils
    module, 289
angr.analyses.cdg
    module, 289
angr.analyses.cfg
    module, 290
angr.analyses.cfg.cfb
    module, 304
angr.analyses.cfg.cfg
    module, 306
angr.analyses.cfg.cfg_arch_options
    module, 307
angr.analyses.cfg.cfg_base
    module, 307
angr.analyses.cfg.cfg_emulated
    module, 310
angr.analyses.cfg.cfg_fast
    module, 315
angr.analyses.cfg.cfg_fast_soot
    module, 325
angr.analyses.cfg.cfg_job_base
    module, 326
angr.analyses.cfg.go_prologue
    module, 326
angr.analyses.cfg.indirect_jump_resolvers
    module, 327
angr.analyses.cfg.indirect_jump_resolvers.aarch64_macho_go
    module, 331
angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_got
    module, 331
angr.analyses.cfg.indirect_jump_resolvers.amd64_pe_iat
    module, 332
angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast
    module, 332
angr.analyses.cfg.indirect_jump_resolvers.const_resolver
    module, 332
angr.analyses.cfg.indirect_jump_resolvers.constant_value_m
    module, 333
angr.analyses.cfg.indirect_jump_resolvers.default_resolver
    module, 333
angr.analyses.cfg.indirect_jump_resolvers.jumptable
    module, 333
angr.analyses.cfg.indirect_jump_resolvers.memload_resolver
    module, 338
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast
    module, 338
angr.analyses.cfg.indirect_jump_resolvers.mips_elf_got
    module, 339
angr.analyses.cfg.indirect_jump_resolvers.propagator_utils
    module, 339
angr.analyses.cfg.indirect_jump_resolvers.resolver
    module, 340

angr.analyses.cfg.indirect_jump_resolvers.syscall_resolver
module, 340
angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt
module, 341
angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat
module, 341
angr.analyses.cfg.meta_structs
module, 341
angr.analyses.cfg.pe_msvc_eh_structs
module, 342
angr.analyses.cfg_slice_to_sink
module, 346
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink
module, 348
angr.analyses.cfg_slice_to_sink.graph
module, 350
angr.analyses.cfg_slice_to_sink.transitions
module, 351
angr.analyses.class_identifier
module, 351
angr.analyses.code_tagging
module, 351
angr.analyses.codecave
module, 352
angr.analyses.complete_calling_conventions
module, 353
angr.analyses.congruency_check
module, 354
angr.analyses.data_dep
module, 355
angr.analyses.data_dep.data_dependency_analysis
module, 358
angr.analyses.data_dep.dep_nodes
module, 359
angr.analyses.data_dep.sim_act_location
module, 361
angr.analyses.datagraph_meta
module, 361
angr.analyses.ddg
module, 361
angr.analyses.decompiler
module, 367
angr.analyses.decompiler.ail_simplifier
module, 396
angr.analyses.decompiler.ailgraph_walker
module, 398
angr.analyses.decompiler.block_io_finder
module, 398
angr.analyses.decompiler.block_similarity
module, 398
angr.analyses.decompiler.block_simplifier
module, 399
angr.analyses.decompiler.block_walkers
module, 401
angr.analyses.decompiler.callsite_maker
module, 401
angr.analyses.decompiler.ccall_rewriters
module, 402
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls
module, 402
angr.analyses.decompiler.ccall_rewriters.arm_ccalls
module, 402
angr.analyses.decompiler.ccall_rewriters.rewriter_base
module, 402
angr.analyses.decompiler.ccall_rewriters.x86_ccalls
module, 403
angr.analyses.decompiler.clinic
module, 403
angr.analyses.decompiler.condition_processor
module, 408
angr.analyses.decompiler.counters
module, 410
angr.analyses.decompiler.counters.boolean_counter
module, 411
angr.analyses.decompiler.counters.call_counter
module, 411
angr.analyses.decompiler.counters.expression_counters
module, 412
angr.analyses.decompiler.counters.seq_cf_structure_counter
module, 412
angr.analyses.decompiler.decompilation_cache
module, 412
angr.analyses.decompiler.decompilation_options
module, 413
angr.analyses.decompiler.decompiler
module, 414
angr.analyses.decompiler.dephication
module, 417
angr.analyses.decompiler.dephication.dephication_base
module, 418
angr.analyses.decompiler.dephication.graph_dephication
module, 419
angr.analyses.decompiler.dephication.graph_rewriting
module, 419
angr.analyses.decompiler.dephication.graph_vvar_mapping
module, 419
angr.analyses.decompiler.dephication.rewriting_engine
module, 419
angr.analyses.decompiler.dephication.seqnode_dephication
module, 420
angr.analyses.decompiler.dirty_rewriters
module, 421
angr.analyses.decompiler.dirty_rewriters.amd64_dirty
module, 421
angr.analyses.decompiler.dirty_rewriters.rewriter_base
module, 421
angr.analyses.decompiler.edits
module, 421

angr.analyses.decompiler.edits.cache
module, 435
angr.analyses.decompiler.edits.errors
module, 437
angr.analyses.decompiler.edits.hooks
module, 438
angr.analyses.decompiler.edits.ops
module, 442
angr.analyses.decompiler.edits.resolve
module, 446
angr.analyses.decompiler.edits.results
module, 449
angr.analyses.decompiler.empty_node_remover
module, 450
angr.analyses.decompiler.expression_narrower
module, 451
angr.analyses.decompiler.goto_manager
module, 452
angr.analyses.decompiler.jump_target_collector
module, 453
angr.analyses.decompiler.jumptable_entry_condition_analyser
module, 453
angr.analyses.decompiler.known_patterns
module, 453
angr.analyses.decompiler.known_patterns.apply
module, 485
angr.analyses.decompiler.known_patterns.block_split
module, 486
angr.analyses.decompiler.known_patterns.containing_memset
module, 487
angr.analyses.decompiler.known_patterns.context
module, 487
angr.analyses.decompiler.known_patterns.ctype_tables
module, 489
angr.analyses.decompiler.known_patterns.dsl
module, 490
angr.analyses.decompiler.known_patterns.finder
module, 507
angr.analyses.decompiler.known_patterns.gating
module, 510
angr.analyses.decompiler.known_patterns.generator
module, 515
angr.analyses.decompiler.known_patterns.kernel_api
module, 516
angr.analyses.decompiler.known_patterns.layouts
module, 517
angr.analyses.decompiler.known_patterns.libm_bits
module, 518
angr.analyses.decompiler.known_patterns.linked_list
module, 518
angr.analyses.decompiler.known_patterns.pattern
module, 519
angr.analyses.decompiler.known_patterns.posix_macros
module, 523
angr.analyses.decompiler.known_patterns.protobuf_hasbits
module, 524
angr.analyses.decompiler.known_patterns.registry
module, 524
angr.analyses.decompiler.known_patterns.std_string_cstr
module, 526
angr.analyses.decompiler.known_patterns.std_string_dtor
module, 526
angr.analyses.decompiler.known_patterns.std_string_interna
module, 527
angr.analyses.decompiler.known_patterns.std_string_length
module, 527
angr.analyses.decompiler.known_patterns.std_swap
module, 527
angr.analyses.decompiler.known_patterns.std_vector_size
module, 528
angr.analyses.decompiler.known_patterns.stl_accessors2
module, 530
angr.analyses.decompiler.known_patterns.stl_containers
module, 531
angr.analyses.decompiler.known_patterns.templates
module, 531
angr.analyses.decompiler.known_patterns.vector_claims
module, 533
angr.analyses.decompiler.known_patterns.vector_math
module, 534
angr.analyses.decompiler.known_patterns.wdk_shared_data
module, 534
angr.analyses.decompiler.label_collector
module, 535
angr.analyses.decompiler.node_replacer
module, 535
angr.analyses.decompiler.notes
module, 535
angr.analyses.decompiler.notes.decompilation_note
module, 538
angr.analyses.decompiler.notes.deobfuscated_strings
module, 539
angr.analyses.decompiler.optimization_pass_registry
module, 540
angr.analyses.decompiler.optimization_passes
module, 540
angr.analyses.decompiler.optimization_passes.base_ptr_save
module, 559
angr.analyses.decompiler.optimization_passes.call_stmt_rew
module, 560
angr.analyses.decompiler.optimization_passes.code_motion
module, 560
angr.analyses.decompiler.optimization_passes.condition_cor
module, 562
angr.analyses.decompiler.optimization_passes.const_derefs
module, 562
angr.analyses.decompiler.optimization_passes.const_prop_re
module, 563

angr.analyses.decompiler.optimization_passes.cross_jump_reverter module, 564
angr.analyses.decompiler.optimization_passes.duplication module, 565
angr.analyses.decompiler.optimization_passes.expr_op_swapper module, 565
angr.analyses.decompiler.optimization_passes.flip_boolean_cmp module, 565
angr.analyses.decompiler.optimization_passes.inline_small_functions module, 566
angr.analyses.decompiler.optimization_passes.lazy_simplifier module, 568
angr.analyses.decompiler.optimization_passes.multi_simplifier module, 570
angr.analyses.decompiler.optimization_passes.optimization_pass module, 571
angr.analyses.decompiler.optimization_passes.optimization_pass_stage module, 572
angr.analyses.decompiler.optimization_passes.register_save_simplifier module, 572
angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier module, 573
angr.analyses.decompiler.optimization_passes.ret_deduplicator module, 574
angr.analyses.decompiler.optimization_passes.return_duplicator_base module, 574
angr.analyses.decompiler.optimization_passes.return_duplicator_high module, 575
angr.analyses.decompiler.optimization_passes.return_duplicator_low module, 576
angr.analyses.decompiler.optimization_passes.stack_canary_simplifier module, 577
angr.analyses.decompiler.optimization_passes.static_vvar_remover module, 577
angr.analyses.decompiler.optimization_passes.switch_default_case_normalizer module, 578
angr.analyses.decompiler.optimization_passes.switch_reused_entry_obj_loader module, 579
angr.analyses.decompiler.optimization_passes.tag_slicer module, 581
angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier module, 581
angr.analyses.decompiler.optimization_passes.x86_gcc_getpc module, 582
angr.analyses.decompiler.peephole_optimizations module, 583
angr.analyses.decompiler.peephole_optimizations.a_div_const_to_mull module, 583
angr.analyses.decompiler.peephole_optimizations.a_mul_const_to_lea module, 584
angr.analyses.decompiler.peephole_optimizations.a_mul_const_to_shl module, 585
angr.analyses.decompiler.peephole_optimizations.a_shl_const_to_mul module, 586
angr.analyses.decompiler.peephole_optimizations.a_sub_a_div_const_to_mull module, 586
angr.analyses.decompiler.peephole_optimizations.a_sub_a_shr_const_to_mull module, 589
angr.analyses.decompiler.peephole_optimizations.a_sub_a_sub_const_to_mull module, 590
angr.analyses.decompiler.peephole_optimizations.arm_cmpf module, 590
angr.analyses.decompiler.peephole_optimizations.base module, 591
angr.analyses.decompiler.peephole_optimizations.basepointer_saver module, 591
angr.analyses.decompiler.peephole_optimizations.bitwise_and_zero module, 592
angr.analyses.decompiler.peephole_optimizations.bitwise_or_zero module, 592
angr.analyses.decompiler.peephole_optimizations.bswap module, 593
angr.analyses.decompiler.peephole_optimizations.constant_derefs module, 594
angr.analyses.decompiler.peephole_optimizations.constant_shifts module, 594
angr.analyses.decompiler.peephole_optimizations.eager_eval module, 596
angr.analyses.decompiler.peephole_optimizations.invert_negated_logical module, 597
angr.analyses.decompiler.peephole_optimizations.mod_simplifier module, 597
angr.analyses.decompiler.peephole_optimizations.mull_const_to_lea module, 598
angr.analyses.decompiler.peephole_optimizations.mull_const_to_shl module, 598
angr.analyses.decompiler.peephole_optimizations.not_sub module, 598
angr.analyses.decompiler.peephole_optimizations.one_sub_bool module, 600
angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite module, 600
angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_regression module, 600
angr.analyses.decompiler.peephole_optimizations.shl_const_to_mul module, 601
angr.analyses.decompiler.peephole_optimizations.sub_a_div_const_to_mull module, 601
angr.analyses.decompiler.peephole_optimizations.sub_a_shr_const_to_mull module, 601
angr.analyses.decompiler.peephole_optimizations.sub_a_sub_const_to_mull module, 602
angr.analyses.decompiler.peephole_optimizations.x86_cmpf module, 602
angr.analyses.decompiler.peephole_optimizations.xor_to_not module, 605

angr.analyses.decompiler.peephole_optimizations.base_analyses.decompiler.peephole_optimizations.remove_con
module, 605
angr.analyses.decompiler.peephole_optimizations.bitwise_insects
module, 605
angr.analyses.decompiler.peephole_optimizations.bitwise_to_logical_or
module, 605
angr.analyses.decompiler.peephole_optimizations.bool_expression
module, 606
angr.analyses.decompiler.peephole_optimizations.bswap
module, 606
angr.analyses.decompiler.peephole_optimizations.bt_analyses_degree
module, 606
angr.analyses.decompiler.peephole_optimizations.cas_analyses.decompiler.peephole_optimizations.remove_red
module, 607
angr.analyses.decompiler.peephole_optimizations.cmp_analyses_decompiler.peephole_optimizations.remove_red
module, 608
angr.analyses.decompiler.peephole_optimizations.cmp_analyses.decompiler.peephole_optimizations.remove_red
module, 608
angr.analyses.decompiler.peephole_optimizations.cmp_analyses.decompiler.peephole_optimizations.remove_red
module, 609
angr.analyses.decompiler.peephole_optimizations.coalesce_adjacent_shrs.peephole_optimizations.remove_red
module, 609
angr.analyses.decompiler.peephole_optimizations.coalesce_same_cascading_ifs
module, 609
angr.analyses.decompiler.peephole_optimizations.conan_analyses.decompiler.peephole_optimizations.remove_red
module, 610
angr.analyses.decompiler.peephole_optimizations.constant_derefs
module, 610
angr.analyses.decompiler.peephole_optimizations.conv_analyses0_decompiler.peephole_optimizations.rewrite_bi
module, 610
angr.analyses.decompiler.peephole_optimizations.conv_shl_shr
module, 610
angr.analyses.decompiler.peephole_optimizations.engen_eval
module, 611
angr.analyses.decompiler.peephole_optimizations.evaluate_const_conversions
module, 611
angr.analyses.decompiler.peephole_optimizations.extended_byte_and_mask.peephole_optimizations.rol_ror
module, 612
angr.analyses.decompiler.peephole_optimizations.inlined_strcpy
module, 612
angr.analyses.decompiler.peephole_optimizations.inlined_strcpy_consolidation
module, 612
angr.analyses.decompiler.peephole_optimizations.invert_negated_logical_conjunction_disjunctions.simplify_p
module, 612
angr.analyses.decompiler.peephole_optimizations.known_bytes_decompiler.peephole_optimizations.single_bit
module, 612
angr.analyses.decompiler.peephole_optimizations.modulo_simplifier
module, 613
angr.analyses.decompiler.peephole_optimizations.one_subbool
module, 613
angr.analyses.decompiler.peephole_optimizations.optimized_div_simplifier
module, 613
angr.analyses.decompiler.peephole_optimizations.remove_cascading_conversions
module, 614
angr.analyses.decompiler.peephole_optimizations.remove_cor
module, 614
angr.analyses.decompiler.peephole_optimizations.remove_cxx
module, 614
angr.analyses.decompiler.peephole_optimizations.remove_emp
module, 614
angr.analyses.decompiler.peephole_optimizations.remove_noc
module, 615
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 615
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 615
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 616
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 616
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 616
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 616
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 617
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 617
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 617
angr.analyses.decompiler.peephole_optimizations.remove_red
module, 618
angr.analyses.decompiler.peephole_optimizations.rewrite_bi
module, 618
angr.analyses.decompiler.peephole_optimizations.rewrite_co
module, 618
angr.analyses.decompiler.peephole_optimizations.rewrite_cx
module, 619
angr.analyses.decompiler.peephole_optimizations.rewrite_mi
module, 619
angr.analyses.decompiler.peephole_optimizations.rol_ror
module, 619
angr.analyses.decompiler.peephole_optimizations.sar_to_sig
module, 620
angr.analyses.decompiler.peephole_optimizations.shl_to_mul
module, 620
angr.analyses.decompiler.peephole_optimizations.simplify_p
module, 620
angr.analyses.decompiler.peephole_optimizations.single_bit
module, 620
angr.analyses.decompiler.peephole_optimizations.single_bit
module, 621
angr.analyses.decompiler.peephole_optimizations.tidy_stack
module, 621
angr.analyses.decompiler.peephole_optimizations.utils
module, 621
angr.analyses.decompiler.peephole_optimizations.remove_cascading_conversions
module, 621

angr.analyses.decompiler.presets.basic
    module, 622
angr.analyses.decompiler.presets.fast
    module, 622
angr.analyses.decompiler.presets.full
    module, 622
angr.analyses.decompiler.presets.malware
    module, 622
angr.analyses.decompiler.presets.preset
    module, 622
angr.analyses.decompiler.redundant_label_remover
    module, 623
angr.analyses.decompiler.region_identifier
    module, 623
angr.analyses.decompiler.region_overlay
    module, 623
angr.analyses.decompiler.region_simplifiers
    module, 633
angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer
    module, 633
angr.analyses.decompiler.region_simplifiers.cascading_ifs
    module, 633
angr.analyses.decompiler.region_simplifiers.expression_narrower
    module, 634
angr.analyses.decompiler.region_simplifiers.goto
    module, 637
angr.analyses.decompiler.region_simplifiers.if_
    module, 638
angr.analyses.decompiler.region_simplifiers.ifelse
    module, 638
angr.analyses.decompiler.region_simplifiers.loop
    module, 638
angr.analyses.decompiler.region_simplifiers.node_duplicator
    module, 638
angr.analyses.decompiler.region_simplifiers.region_simplifier
    module, 638
angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier
    module, 639
angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier
    module, 641
angr.analyses.decompiler.region_walker
    module, 642
angr.analyses.decompiler.return_maker
    module, 642
angr.analyses.decompiler.semantic_naming
    module, 642
angr.analyses.decompiler.semantic_naming.array_index_analysis
    module, 647
angr.analyses.decompiler.semantic_naming.boolean_naming
    module, 647
angr.analyses.decompiler.semantic_naming.call_arg_naming
    module, 648
angr.analyses.decompiler.semantic_naming.naming_base
    module, 648
angr.analyses.decompiler.semantic_naming.orchestrator
    module, 650
angr.analyses.decompiler.semantic_naming.pointer_naming
    module, 651
angr.analyses.decompiler.semantic_naming.region_loop_count
    module, 651
angr.analyses.decompiler.semantic_naming.size_naming
    module, 652
angr.analyses.decompiler.seq_to_blocks
    module, 652
angr.analyses.decompiler.sequence_walker
    module, 652
angr.analyses.decompiler.ssailification
    module, 652
angr.analyses.decompiler.ssailification.consts
    module, 653
angr.analyses.decompiler.ssailification.rewriting
    module, 653
angr.analyses.decompiler.ssailification.rewriting_engine
    module, 654
angr.analyses.decompiler.ssailification.rewriting_state
    module, 654
angr.analyses.decompiler.ssailification.ssailification
    module, 655
angr.analyses.decompiler.ssailification.traversal
    module, 655
angr.analyses.decompiler.ssailification.traversal_engine
    module, 655
angr.analyses.decompiler.ssailification.traversal_state
    module, 658
angr.analyses.decompiler.stack_item
    module, 659
angr.analyses.decompiler.stackarg_offset_manager
    module, 660
angr.analyses.decompiler.stl_field_accessors
    module, 661
angr.analyses.decompiler.structured_codegen
    module, 661
angr.analyses.decompiler.structured_codegen.base
    module, 668
angr.analyses.decompiler.structured_codegen.c
    module, 670
angr.analyses.decompiler.structured_codegen.c_serialize
    module, 688
angr.analyses.decompiler.structured_codegen.dummy
    module, 691
angr.analyses.decompiler.structured_codegen.dwarf_import
    module, 691
angr.analyses.decompiler.structured_codegen.rust
    module, 691
angr.analyses.decompiler.structurer_nodes
    module, 710
angr.analyses.decompiler.structuring
    module, 714

angr.analyses.decompiler.structuring.dream
    module, 716
angr.analyses.decompiler.structuring.phoenix
    module, 717
angr.analyses.decompiler.structuring.recursive_structurer
    module, 718
angr.analyses.decompiler.structuring.sailr
    module, 718
angr.analyses.decompiler.structuring.structurer_base
    module, 719
angr.analyses.decompiler.utils
    module, 720
angr.analyses.decompiler.variable_map
    module, 727
angr.analyses.deobfuscator
    module, 731
angr.analyses.deobfuscator.api_obf_finder
    module, 734
angr.analyses.deobfuscator.api_obf_peephole_optimizer
    module, 735
angr.analyses.deobfuscator.api_obf_type2_finder
    module, 736
angr.analyses.deobfuscator.data_transformation_graph_analyzer
    module, 737
angr.analyses.deobfuscator.hash_lookup_api_deobfuscator
    module, 737
angr.analyses.deobfuscator.irsb_reg_collector
    module, 737
angr.analyses.deobfuscator.scope_ops_analyzer
    module, 738
angr.analyses.deobfuscator.string_obf_finder
    module, 738
angr.analyses.deobfuscator.string_obf_opt_passes
    module, 739
angr.analyses.deobfuscator.string_obf_peephole_optimizer
    module, 739
angr.analyses.disassembly
    module, 739
angr.analyses.disassembly_utils
    module, 743
angr.analyses.dominance_frontier
    module, 743
angr.analyses.fcp
    module, 744
angr.analyses.fcp.fcp
    module, 744
angr.analyses.find_objects_static
    module, 746
angr.analyses.flirt
    module, 747
angr.analyses.flirt.consts
    module, 751
angr.analyses.flirt.flirt
    module, 755
angr.analyses.flirt.flirt_function
    module, 756
angr.analyses.flirt.flirt_matcher
    module, 756
angr.analyses.flirt.flirt_module
    module, 757
angr.analyses.flirt.flirt_node
    module, 757
angr.analyses.flirt.flirt_sig
    module, 758
angr.analyses.flirt.flirt_utils
    module, 761
angr.analyses.forward_analysis
    module, 761
angr.analyses.forward_analysis.forward_analysis
    module, 763
angr.analyses.forward_analysis.job_info
    module, 764
angr.analyses.forward_analysis.visitors
    module, 765
angr.analyses.forward_analysis.visitors.call_graph
    module, 766
angr.analyses.forward_analysis.visitors.function_graph
    module, 766
angr.analyses.forward_analysis.visitors.graph
    module, 766
angr.analyses.forward_analysis.visitors.loop
    module, 768
angr.analyses.forward_analysis.visitors.single_node_graph
    module, 769
angr.analyses.identifier
    module, 769
angr.analyses.identifier.custom_callable
    module, 770
angr.analyses.identifier.errors
    module, 770
angr.analyses.identifier.func
    module, 771
angr.analyses.identifier.functions
    module, 771
angr.analyses.identifier.functions.atoi
    module, 772
angr.analyses.identifier.functions.based_atoi
    module, 772
angr.analyses.identifier.functions.fdprintf
    module, 772
angr.analyses.identifier.functions.free
    module, 773
angr.analyses.identifier.functions.int2str
    module, 773
angr.analyses.identifier.functions.malloc
    module, 774
angr.analyses.identifier.functions.memcmp
    module, 775

angr.analyses.identifier.functions.memcpy
module, 775
angr.analyses.identifier.functions.memset
module, 776
angr.analyses.identifier.functions.printf
module, 776
angr.analyses.identifier.functions.recv_until
module, 777
angr.analyses.identifier.functions.skip_calloc
module, 778
angr.analyses.identifier.functions.skip_realloc
module, 778
angr.analyses.identifier.functions.skip_recv_n
module, 778
angr.analyses.identifier.functions.snprintf
module, 779
angr.analyses.identifier.functions.sprintf
module, 780
angr.analyses.identifier.functions.strcasecmp
module, 780
angr.analyses.identifier.functions.strcmp
module, 781
angr.analyses.identifier.functions.strcpy
module, 782
angr.analyses.identifier.functions.strlen
module, 782
angr.analyses.identifier.functions.strncmp
module, 783
angr.analyses.identifier.functions.strncpy
module, 784
angr.analyses.identifier.functions.strtol
module, 784
angr.analyses.identifier.identify
module, 784
angr.analyses.identifier.runner
module, 785
angr.analyses.init_finder
module, 786
angr.analyses.language_detector
module, 786
angr.analyses.loop_analysis
module, 787
angr.analyses.loop_analysis.loop_analysis
module, 787
angr.analyses.loop_unroller
module, 790
angr.analyses.loop_unroller.loop_unroller
module, 791
angr.analyses.loopfinder
module, 791
angr.analyses.outliner
module, 791
angr.analyses.outliner.outliner
module, 792
angr.analyses.patchfinder
module, 792
angr.analyses.pathfinder
module, 794
angr.analyses.propagator
module, 796
angr.analyses.propagator.engine_base
module, 796
angr.analyses.propagator.engine_vex
module, 797
angr.analyses.propagator.propagator
module, 797
angr.analyses.propagator.top_checker_mixin
module, 797
angr.analyses.propagator.values
module, 798
angr.analyses.propagator.vex_vars
module, 798
angr.analyses.proximity_graph
module, 798
angr.analyses.purity
module, 800
angr.analyses.purity.analysis
module, 801
angr.analyses.purity.engine
module, 801
angr.analyses.reaching_definitions
module, 804
angr.analyses.reaching_definitions.call_trace
module, 834
angr.analyses.reaching_definitions.dep_graph
module, 835
angr.analyses.reaching_definitions.engine_ail
module, 838
angr.analyses.reaching_definitions.engine_vex
module, 838
angr.analyses.reaching_definitions.external_codeloc
module, 838
angr.analyses.reaching_definitions.function_handler
module, 838
angr.analyses.reaching_definitions.function_handler_librar
module, 845
angr.analyses.reaching_definitions.function_handler_librar
module, 846
angr.analyses.reaching_definitions.function_handler_librar
module, 850
angr.analyses.reaching_definitions.function_handler_librar
module, 852
angr.analyses.reaching_definitions.function_handler_librar
module, 855
angr.analyses.reaching_definitions.heap_allocator
module, 857
angr.analyses.reaching_definitions.rd_initializer
module, 857

angr.analyses.reaching_definitions.rd_state
    module, 859
angr.analyses.reaching_definitions.reaching_definitions
    module, 867
angr.analyses.reaching_definitions.subject
    module, 869
angr.analyses.reassembler
    module, 870
angr.analyses.s_liveness
    module, 878
angr.analyses.s_propagator
    module, 879
angr.analyses.s_reaching_definitions
    module, 880
angr.analyses.s_reaching_definitions.s_rda_model
    module, 885
angr.analyses.s_reaching_definitions.s_rda_view
    module, 888
angr.analyses.s_reaching_definitions.s_reaching_definitions
    module, 891
angr.analyses.smc
    module, 891
angr.analyses.soot_class_hierarchy
    module, 893
angr.analyses.stack_pointer_tracker
    module, 893
angr.analyses.static_hooker
    module, 896
angr.analyses.typehoon
    module, 896
angr.analyses.typehoon.dfa
    module, 898
angr.analyses.typehoon.simple_solver
    module, 899
angr.analyses.typehoon.translator
    module, 905
angr.analyses.typehoon.typeconsts
    module, 906
angr.analyses.typehoon.typehoon
    module, 913
angr.analyses.typehoon.typevars
    module, 914
angr.analyses.typehoon.variance
    module, 920
angr.analyses.unpacker
    module, 921
angr.analyses.unpacker.obfuscation_detector
    module, 921
angr.analyses.unpacker.packing_detector
    module, 922
angr.analyses.variable_recovery
    module, 922
angr.analyses.variable_recovery.annotations
    module, 923
angr.analyses.variable_recovery.engine_ail
    module, 923
angr.analyses.variable_recovery.engine_base
    module, 924
angr.analyses.variable_recovery.engine_vex
    module, 925
angr.analyses.variable_recovery.irsb_scanner
    module, 925
angr.analyses.variable_recovery.variable_recovery
    module, 925
angr.analyses.variable_recovery.variable_recovery_base
    module, 926
angr.analyses.variable_recovery.variable_recovery_fast
    module, 930
angr.analyses.veritesting
    module, 930
angr.analyses.vfg
    module, 932
angr.analyses.vsa_ddg
    module, 936
angr.analyses.vtable
    module, 937
angr.analyses.xrefs
    module, 937
angr.angrdb
    module, 938
angr.angrdb.db
    module, 940
angr.angrdb.models
    module, 942
angr.angrdb.serializers
    module, 948
angr.angrdb.serializers.bookmarks
    module, 949
angr.angrdb.serializers.callgraph
    module, 950
angr.angrdb.serializers.cfg_model
    module, 951
angr.angrdb.serializers.comments
    module, 951
angr.angrdb.serializers.funcs
    module, 951
angr.angrdb.serializers.kb
    module, 952
angr.angrdb.serializers.labels
    module, 952
angr.angrdb.serializers.loader
    module, 953
angr.angrdb.serializers.structured_code
    module, 954
angr.angrdb.serializers.variables
    module, 955
angr.angrdb.serializers.xrefs
    module, 956

angr.annocfg
    module, 957
angr.blade
    module, 958
angr.block
    module, 959
angr.callable
    module, 961
angr.calling_conventions
    module, 962
angr.code_location
    module, 985
angr.codenode
    module, 987
angr.concretization_strategies
    module, 988
angr.concretization_strategies.any
    module, 990
angr.concretization_strategies.any_named
    module, 990
angr.concretization_strategies.base
    module, 990
angr.concretization_strategies.controlled_data
    module, 991
angr.concretization_strategies.eval
    module, 991
angr.concretization_strategies.logging
    module, 991
angr.concretization_strategies.max
    module, 991
angr.concretization_strategies.nonzero
    module, 992
angr.concretization_strategies.nonzero_range
    module, 992
angr.concretization_strategies.norepeats
    module, 992
angr.concretization_strategies.norepeats_range
    module, 992
angr.concretization_strategies.range
    module, 992
angr.concretization_strategies.signed_add
    module, 992
angr.concretization_strategies.single
    module, 992
angr.concretization_strategies.solutions
    module, 992
angr.concretization_strategies.unlimited_range
    module, 993
angr.distributed
    module, 993
angr.distributed.server
    module, 994
angr.distributed.worker
    module, 994
angr.emulator
    module, 995
angr.engines
    module, 996
angr.engines.ail
    module, 1001
angr.engines.ail.callstack
    module, 1003
angr.engines.ail.engine_light
    module, 1003
angr.engines.ail.engine_successors
    module, 1004
angr.engines.ail.setup
    module, 1004
angr.engines.engine
    module, 1004
angr.engines.failure
    module, 1005
angr.engines.hook
    module, 1005
angr.engines.icicle
    module, 1005
angr.engines.light
    module, 1005
angr.engines.light.data
    module, 1009
angr.engines.light.engine
    module, 1010
angr.engines.pcode
    module, 1014
angr.engines.pcode.behavior
    module, 1014
angr.engines.pcode.cc
    module, 1034
angr.engines.pcode.emulate
    module, 1036
angr.engines.pcode.engine
    module, 1036
angr.engines.pcode.lifter
    module, 1037
angr.engines.procedure
    module, 1046
angr.engines.soot
    module, 1046
angr.engines.soot.engine
    module, 1047
angr.engines.soot.exceptions
    module, 1047
angr.engines.soot.expressions
    module, 1047
angr.engines.soot.expressions.arrayref
    module, 1050
angr.engines.soot.expressions.base
    module, 1050

angr.engines.soot.expressions.binop
    module, 1050
angr.engines.soot.expressions.cast
    module, 1050
angr.engines.soot.expressions.condition
    module, 1050
angr.engines.soot.expressions.constants
    module, 1050
angr.engines.soot.expressions.instancefieldref
    module, 1051
angr.engines.soot.expressions.instanceOf
    module, 1051
angr.engines.soot.expressions.invoke
    module, 1051
angr.engines.soot.expressions.length
    module, 1051
angr.engines.soot.expressions.local
    module, 1051
angr.engines.soot.expressions.new
    module, 1052
angr.engines.soot.expressions.newArray
    module, 1052
angr.engines.soot.expressions.newMultiArray
    module, 1052
angr.engines.soot.expressions.paramref
    module, 1052
angr.engines.soot.expressions.phi
    module, 1052
angr.engines.soot.expressions.staticfieldref
    module, 1052
angr.engines.soot.expressions.thisref
    module, 1052
angr.engines.soot.expressions.unsupported
    module, 1052
angr.engines.soot.field_dispatcher
    module, 1052
angr.engines.soot.method_dispatcher
    module, 1053
angr.engines.soot.statements
    module, 1053
angr.engines.soot.statements.assign
    module, 1054
angr.engines.soot.statements.base
    module, 1054
angr.engines.soot.statements.goto
    module, 1054
angr.engines.soot.statements.identity
    module, 1054
angr.engines.soot.statements.if_
    module, 1054
angr.engines.soot.statements.invoke
    module, 1054
angr.engines.soot.statements.return_
    module, 1054
angr.engines.soot.statements.switch
    module, 1055
angr.engines.soot.statements.throw
    module, 1055
angr.engines.soot.values
    module, 1055
angr.engines.soot.values.arrayref
    module, 1059
angr.engines.soot.values.base
    module, 1060
angr.engines.soot.values.constants
    module, 1060
angr.engines.soot.values.instancefieldref
    module, 1060
angr.engines.soot.values.local
    module, 1060
angr.engines.soot.values.paramref
    module, 1061
angr.engines.soot.values.staticfieldref
    module, 1061
angr.engines.soot.values.strref
    module, 1061
angr.engines.soot.values.thisref
    module, 1062
angr.engines.successors
    module, 1063
angr.engines.syscall
    module, 1065
angr.engines.unicorn
    module, 1065
angr.engines.vex
    module, 1065
angr.engines.vex.claripy
    module, 1068
angr.engines.vex.claripy.ccall
    module, 1068
angr.engines.vex.claripy.datalayer
    module, 1073
angr.engines.vex.claripy.irop
    module, 1073
angr.engines.vex.heavy
    module, 1074
angr.engines.vex.heavy.actions
    module, 1075
angr.engines.vex.heavy.concretizers
    module, 1075
angr.engines.vex.heavy.dirty
    module, 1076
angr.engines.vex.heavy.heavy
    module, 1078
angr.engines.vex.heavy.inspect
    module, 1078
angr.engines.vex.heavy.resilience
    module, 1079

angr.engines.vex.heavy.super_fastpath
module, 1079
angr.engines.vex.lifter
module, 1079
angr.engines.vex.light
module, 1080
angr.engines.vex.light.light
module, 1080
angr.engines.vex.light.resilience
module, 1080
angr.engines.vex.light.slicing
module, 1081
angr.errors
module, 1081
angr.exploration_techniques
module, 1087
angr.exploration_techniques.base
module, 1097
angr.exploration_techniques.bucketizer
module, 1098
angr.exploration_techniques.common
module, 1098
angr.exploration_techniques.dfs
module, 1099
angr.exploration_techniques.director
module, 1099
angr.exploration_techniques.driller_core
module, 1101
angr.exploration_techniques.explorer
module, 1102
angr.exploration_techniques.lengthlimiter
module, 1102
angr.exploration_techniques.local_loop_seer
module, 1102
angr.exploration_techniques.loop_seer
module, 1102
angr.exploration_techniques.manual_mergepoint
module, 1103
angr.exploration_techniques.memory_watcher
module, 1103
angr.exploration_techniques.oppologist
module, 1103
angr.exploration_techniques.slicecutor
module, 1104
angr.exploration_techniques.spiller
module, 1104
angr.exploration_techniques.spiller_db
module, 1105
angr.exploration_techniques.stochastic
module, 1106
angr.exploration_techniques.stub_stasher
module, 1106
angr.exploration_techniques.suggestions
module, 1106
angr.exploration_techniques.tech_builder
module, 1106
angr.exploration_techniques.threading
module, 1107
angr.exploration_techniques.timeout
module, 1107
angr.exploration_techniques.tracer
module, 1107
angr.exploration_techniques.unique
module, 1109
angr.exploration_techniques.veritesting
module, 1109
angr.factory
module, 1109
angr.flirt
module, 1113
angr.flirt.build_sig
module, 1114
angr.keyed_region
module, 1115
angr.knowledge_base
module, 1118
angr.knowledge_plugins
module, 1120
angr.knowledge_plugins.bookmarks
module, 1149
angr.knowledge_plugins.callsite_prototypes
module, 1150
angr.knowledge_plugins.cfg
module, 1152
angr.knowledge_plugins.cfg.block_id
module, 1165
angr.knowledge_plugins.cfg.cfg_manager
module, 1165
angr.knowledge_plugins.cfg.cfg_model
module, 1166
angr.knowledge_plugins.cfg.cfg_node
module, 1173
angr.knowledge_plugins.cfg.indirect_jump
module, 1175
angr.knowledge_plugins.cfg.memory_data
module, 1177
angr.knowledge_plugins.cfg.spilling_cfg
module, 1179
angr.knowledge_plugins.cfg.spilling_digraph
module, 1187
angr.knowledge_plugins.cfg.types
module, 1189
angr.knowledge_plugins.comments
module, 1189
angr.knowledge_plugins.custom_strings
module, 1191
angr.knowledge_plugins.data
module, 1191

angr.knowledge_plugins.debug_variables
    module, 1192
angr.knowledge_plugins.functions
    module, 1194
angr.knowledge_plugins.functions.function
    module, 1209
angr.knowledge_plugins.functions.function_manager
    module, 1218
angr.knowledge_plugins.functions.function_parser
    module, 1227
angr.knowledge_plugins.functions.soot_function
    module, 1228
angr.knowledge_plugins.indirect_jumps
    module, 1228
angr.knowledge_plugins.key_definitions
    module, 1228
angr.knowledge_plugins.key_definitions.atoms
    module, 1241
angr.knowledge_plugins.key_definitions.constants
    module, 1245
angr.knowledge_plugins.key_definitions.definition
    module, 1245
angr.knowledge_plugins.key_definitions.environment
    module, 1247
angr.knowledge_plugins.key_definitions.heap_address
    module, 1248
angr.knowledge_plugins.key_definitions.key_definition
    module, 1248
angr.knowledge_plugins.key_definitions.live_definitions
    module, 1249
angr.knowledge_plugins.key_definitions.liveness
    module, 1257
angr.knowledge_plugins.key_definitions.rd_model
    module, 1259
angr.knowledge_plugins.key_definitions.tag
    module, 1261
angr.knowledge_plugins.key_definitions.undefined
    module, 1262
angr.knowledge_plugins.key_definitions.unknown_size
    module, 1262
angr.knowledge_plugins.key_definitions.uses
    module, 1262
angr.knowledge_plugins.labels
    module, 1264
angr.knowledge_plugins.obfuscations
    module, 1264
angr.knowledge_plugins.patches
    module, 1264
angr.knowledge_plugins.plugin
    module, 1266
angr.knowledge_plugins.propagations
    module, 1266
angr.knowledge_plugins.propagations.prop_value
    module, 1268
angr.knowledge_plugins.propagations.propagation_manager
    module, 1270
angr.knowledge_plugins.propagations.propagation_model
    module, 1271
angr.knowledge_plugins.propagations.states
    module, 1272
angr.knowledge_plugins.rtdb
    module, 1276
angr.knowledge_plugins.rtdb.rtdb
    module, 1277
angr.knowledge_plugins.structured_code
    module, 1278
angr.knowledge_plugins.types
    module, 1280
angr.knowledge_plugins.variables
    module, 1280
angr.knowledge_plugins.variables.spilling_vardict
    module, 1282
angr.knowledge_plugins.variables.variable_access
    module, 1282
angr.knowledge_plugins.variables.variable_manager
    module, 1283
angr.knowledge_plugins.xrefs
    module, 1290
angr.knowledge_plugins.xrefs.xref
    module, 1291
angr.knowledge_plugins.xrefs.xref_manager
    module, 1292
angr.knowledge_plugins.xrefs.xref_types
    module, 1293
angr.llm_client
    module, 1293
angr.llm_models
    module, 1294
angr.mcp
    module, 1295
angr.mcp.edit_tools
    module, 1299
angr.mcp.errors
    module, 1301
angr.mcp.serializers
    module, 1301
angr.mcp.server
    module, 1303
angr.mcp.session
    module, 1307
angr.misc
    module, 1309
angr.misc.ansi
    module, 1314
angr.misc.autoimport
    module, 1314
angr.misc.bug_report
    module, 1315

angr.misc.hookset
    module, 1315
angr.misc.loggers
    module, 1316
angr.misc.picklable_lock
    module, 1317
angr.misc.plugins
    module, 1317
angr.misc.telemetry
    module, 1320
angr.misc.testing
    module, 1322
angr.misc.ux
    module, 1322
angr.procedures
    module, 1322
angr.procedures.advapi32
    module, 1323
angr.procedures.cgc
    module, 1323
angr.procedures.cgc.allocate
    module, 1323
angr.procedures.cgc.deallocate
    module, 1323
angr.procedures.cgc.fdwait
    module, 1323
angr.procedures.cgc.random
    module, 1323
angr.procedures.cgc.receive
    module, 1324
angr.procedures.cgc.transmit
    module, 1324
angr.procedures.definitions
    module, 1324
angr.procedures.definitions.cgc
    module, 1333
angr.procedures.definitions.gnulib
    module, 1333
angr.procedures.definitions.libstdcpp
    module, 1333
angr.procedures.definitions.linux_kernel
    module, 1333
angr.procedures.definitions.linux_loader
    module, 1333
angr.procedures.definitions.macho_libsystem
    module, 1333
angr.procedures.definitions.msvcr
    module, 1333
angr.procedures.definitions.parse_glibc
    module, 1333
angr.procedures.definitions.parse_syscalls_from_local_system
    module, 1334
angr.procedures.definitions.parse_win32json
    module, 1334
angr.procedures.definitions.types_stl
    module, 1334
angr.procedures.glibc
    module, 1336
angr.procedures.glibc.dynamic_loading
    module, 1336
angr.procedures.glibc.scanf
    module, 1336
angr.procedures.glibc.sscanf
    module, 1336
angr.procedures.gnulib
    module, 1336
angr.procedures.gnulib.xalloc_die
    module, 1337
angr.procedures.gnulib.xstrtol_fatal
    module, 1337
angr.procedures.java
    module, 1337
angr.procedures.java.unconstrained
    module, 1337
angr.procedures.java_io
    module, 1337
angr.procedures.java_io.read
    module, 1337
angr.procedures.java_io.write
    module, 1338
angr.procedures.java_jni
    module, 1338
angr.procedures.java_jni.array_operations
    module, 1338
angr.procedures.java_jni.class_and_interface_operations
    module, 1340
angr.procedures.java_jni.field_access
    module, 1340
angr.procedures.java_jni.global_and_local_refs
    module, 1341
angr.procedures.java_jni.method_calls
    module, 1342
angr.procedures.java_jni.not_implemented
    module, 1346
angr.procedures.java_jni.object_operations
    module, 1347
angr.procedures.java_jni.string_operations
    module, 1347
angr.procedures.java_jni.version_information
    module, 1347
angr.procedures.java_lang
    module, 1348
angr.procedures.java_lang.character
    module, 1348
angr.procedures.java_lang.double
    module, 1348
angr.procedures.java_lang.exit
    module, 1348

angr.procedures.java_lang.getsimplename
module, 1348
angr.procedures.java_lang.integer
module, 1348
angr.procedures.java_lang.load_library
module, 1349
angr.procedures.java_lang.math
module, 1349
angr.procedures.java_lang.string
module, 1349
angr.procedures.java_lang.stringbuilder
module, 1349
angr.procedures.java_lang.system
module, 1349
angr.procedures.java_util
module, 1349
angr.procedures.java_util.collection
module, 1350
angr.procedures.java_util.iterator
module, 1350
angr.procedures.java_util.list
module, 1350
angr.procedures.java_util.map
module, 1350
angr.procedures.java_util.random
module, 1350
angr.procedures.java_util.scanner_nextline
module, 1351
angr.procedures.libc
module, 1351
angr.procedures.libc.abort
module, 1353
angr.procedures.libc.access
module, 1353
angr.procedures.libc.atoi
module, 1353
angr.procedures.libc.atol
module, 1353
angr.procedures.libc.calloc
module, 1353
angr.procedures.libc.closelog
module, 1353
angr.procedures.libc.err
module, 1353
angr.procedures.libc.error
module, 1353
angr.procedures.libc.exit
module, 1353
angr.procedures.libc.fclose
module, 1354
angr.procedures.libc.feof
module, 1354
angr.procedures.libc.fflush
module, 1354
angr.procedures.libc.fgetc
module, 1354
angr.procedures.libc.fgets
module, 1354
angr.procedures.libc.fopen
module, 1354
angr.procedures.libc.fprintf
module, 1354
angr.procedures.libc.fputc
module, 1355
angr.procedures.libc.fputs
module, 1355
angr.procedures.libc.fread
module, 1355
angr.procedures.libc.free
module, 1355
angr.procedures.libc.fscanf
module, 1355
angr.procedures.libc.fseek
module, 1355
angr.procedures.libc.ftell
module, 1355
angr.procedures.libc.fwrite
module, 1356
angr.procedures.libc.getchar
module, 1356
angr.procedures.libc.getdelim
module, 1356
angr.procedures.libc.getegid
module, 1356
angr.procedures.libc.geteuid
module, 1356
angr.procedures.libc.getgid
module, 1356
angr.procedures.libc.gets
module, 1356
angr.procedures.libc.getuid
module, 1356
angr.procedures.libc.isalnum
module, 1356
angr.procedures.libc.isalpha
module, 1356
angr.procedures.libc.isascii
module, 1357
angr.procedures.libc.isblank
module, 1357
angr.procedures.libc.iscntrl
module, 1357
angr.procedures.libc.isdigit
module, 1357
angr.procedures.libc.isgraph
module, 1357
angr.procedures.libc.islower
module, 1357

angr.procedures.libc.isprint
module, 1357
angr.procedures.libc.ispunct
module, 1357
angr.procedures.libc.isspace
module, 1357
angr.procedures.libc.isupper
module, 1357
angr.procedures.libc.isxdigit
module, 1357
angr.procedures.libc.malloc
module, 1358
angr.procedures.libc.memccpy
module, 1358
angr.procedures.libc.memchr
module, 1358
angr.procedures.libc.memcmp
module, 1358
angr.procedures.libc.memcpy
module, 1358
angr.procedures.libc.memmove
module, 1358
angr.procedures.libc.mempcpy
module, 1358
angr.procedures.libc.memset
module, 1358
angr.procedures.libc.openlog
module, 1358
angr.procedures.libc.perror
module, 1358
angr.procedures.libc.printf
module, 1358
angr.procedures.libc.putchar
module, 1359
angr.procedures.libc.puts
module, 1359
angr.procedures.libc.rand
module, 1359
angr.procedures.libc.realloc
module, 1359
angr.procedures.libc.rewind
module, 1359
angr.procedures.libc.scanf
module, 1359
angr.procedures.libc.setbuf
module, 1359
angr.procedures.libc.setvbuf
module, 1359
angr.procedures.libc.snprintf
module, 1359
angr.procedures.libc.sprintf
module, 1359
angr.procedures.libc.srand
module, 1359
angr.procedures.libc.sscanf
module, 1360
angr.procedures.libc.stpcpy
module, 1360
angr.procedures.libc.strcat
module, 1360
angr.procedures.libc.strchr
module, 1360
angr.procedures.libc.strcmp
module, 1360
angr.procedures.libc.strcoll
module, 1360
angr.procedures.libc.strcpy
module, 1360
angr.procedures.libc.strlen
module, 1360
angr.procedures.libc.strncat
module, 1360
angr.procedures.libc.strncmp
module, 1360
angr.procedures.libc.strncpy
module, 1361
angr.procedures.libc.strnlen
module, 1361
angr.procedures.libc.strstr
module, 1361
angr.procedures.libc.strtok
module, 1361
angr.procedures.libc.strtol
module, 1361
angr.procedures.libc.strtoul
module, 1361
angr.procedures.libc.strxfrm
module, 1362
angr.procedures.libc.system
module, 1362
angr.procedures.libc.time
module, 1362
angr.procedures.libc.tmpnam
module, 1362
angr.procedures.libc.tolower
module, 1362
angr.procedures.libc.toupper
module, 1362
angr.procedures.libc.ungetc
module, 1362
angr.procedures.libc.vsnprintf
module, 1362
angr.procedures.libc.wchar
module, 1362
angr.procedures.libstdcpp
module, 1362
angr.procedures.libstdcpp.std____throw_bad_alloc
module, 1363

angr.procedures.libstdcpp.std____throw_bad_cast
    module, 1363
angr.procedures.libstdcpp.std____throw_length_error
    module, 1363
angr.procedures.libstdcpp.std____throw_logic_error
    module, 1363
angr.procedures.libstdcpp.std__terminate
    module, 1363
angr.procedures.linux_kernel
    module, 1363
angr.procedures.linux_kernel.access
    module, 1364
angr.procedures.linux_kernel.arch_prctl
    module, 1364
angr.procedures.linux_kernel.arm_user_helpers
    module, 1364
angr.procedures.linux_kernel.brk
    module, 1364
angr.procedures.linux_kernel.cwd
    module, 1365
angr.procedures.linux_kernel.fstat
    module, 1365
angr.procedures.linux_kernel.fstat64
    module, 1365
angr.procedures.linux_kernel.futex
    module, 1365
angr.procedures.linux_kernel.getegid
    module, 1365
angr.procedures.linux_kernel.geteuid
    module, 1365
angr.procedures.linux_kernel.getgid
    module, 1365
angr.procedures.linux_kernel.getpid
    module, 1365
angr.procedures.linux_kernel.getrlimit
    module, 1366
angr.procedures.linux_kernel.gettid
    module, 1366
angr.procedures.linux_kernel.getuid
    module, 1366
angr.procedures.linux_kernel.iovec
    module, 1366
angr.procedures.linux_kernel.lseek
    module, 1366
angr.procedures.linux_kernel.mmap
    module, 1366
angr.procedures.linux_kernel.mprotect
    module, 1366
angr.procedures.linux_kernel.munmap
    module, 1366
angr.procedures.linux_kernel.openat
    module, 1367
angr.procedures.linux_kernel.set_tid_address
    module, 1367
angr.procedures.linux_kernel.sigaction
    module, 1367
angr.procedures.linux_kernel.sigprocmask
    module, 1367
angr.procedures.linux_kernel.stat
    module, 1367
angr.procedures.linux_kernel.sysinfo
    module, 1367
angr.procedures.linux_kernel.tgkill
    module, 1367
angr.procedures.linux_kernel.time
    module, 1367
angr.procedures.linux_kernel.uid
    module, 1367
angr.procedures.linux_kernel.uname
    module, 1368
angr.procedures.linux_kernel.unlink
    module, 1368
angr.procedures.linux_kernel.vsyscall
    module, 1368
angr.procedures.linux_loader
    module, 1368
angr.procedures.linux_loader.sim_loader
    module, 1368
angr.procedures.linux_loader.tls
    module, 1368
angr.procedures.msvcr
    module, 1369
angr.procedures.msvcr.fmode
    module, 1369
angr.procedures.ntdll
    module, 1369
angr.procedures.ntdll.exceptions
    module, 1369
angr.procedures.posix
    module, 1369
angr.procedures.posix.accept
    module, 1370
angr.procedures.posix.bind
    module, 1370
angr.procedures.posix.bzero
    module, 1370
angr.procedures.posix.chroot
    module, 1371
angr.procedures.posix.close
    module, 1371
angr.procedures.posix.closedir
    module, 1371
angr.procedures.posix.dup
    module, 1371
angr.procedures.posix.fcntl
    module, 1371
angr.procedures.posix.fdopen
    module, 1371

angr.procedures.posix.fileno
    module, 1371
angr.procedures.posix.fork
    module, 1372
angr.procedures.posix.getenv
    module, 1372
angr.procedures.posix.gethostbyname
    module, 1372
angr.procedures.posix.getpass
    module, 1372
angr.procedures.posix.getsockopt
    module, 1372
angr.procedures.posix.htonl
    module, 1372
angr.procedures.posix.htons
    module, 1372
angr.procedures.posix.inet_ntoa
    module, 1372
angr.procedures.posix.listen
    module, 1373
angr.procedures.posix.mmap
    module, 1373
angr.procedures.posix.open
    module, 1373
angr.procedures.posix.opendir
    module, 1373
angr.procedures.posix.poll
    module, 1373
angr.procedures.posix.pread64
    module, 1373
angr.procedures.posix.pthread
    module, 1373
angr.procedures.posix.pwrite64
    module, 1374
angr.procedures.posix.read
    module, 1374
angr.procedures.posix.readdir
    module, 1374
angr.procedures.posix.recv
    module, 1374
angr.procedures.posix.recvfrom
    module, 1374
angr.procedures.posix.select
    module, 1374
angr.procedures.posix.send
    module, 1374
angr.procedures.posix.setsockopt
    module, 1375
angr.procedures.posix.sigaction
    module, 1375
angr.procedures.posix.sim_time
    module, 1375
angr.procedures.posix.sleep
    module, 1375
angr.procedures.posix.socket
    module, 1375
angr.procedures.posix.strcasecmp
    module, 1375
angr.procedures.posix.strdup
    module, 1375
angr.procedures.posix.strncasecmp
    module, 1375
angr.procedures.posix.strndup
    module, 1375
angr.procedures.posix.strtok_r
    module, 1375
angr.procedures.posix.syslog
    module, 1376
angr.procedures.posix.tz
    module, 1376
angr.procedures.posix.unlink
    module, 1376
angr.procedures.posix.usleep
    module, 1376
angr.procedures.posix.write
    module, 1376
angr.procedures.procedure_dict
    module, 1376
angr.procedures.stubs
    module, 1376
angr.procedures.stubs.b64_decode
    module, 1378
angr.procedures.stubs.caller
    module, 1378
angr.procedures.stubs.CallReturn
    module, 1376
angr.procedures.stubs.crazy_scanf
    module, 1378
angr.procedures.stubs.format_parser
    module, 1378
angr.procedures.stubs.Nop
    module, 1377
angr.procedures.stubs.NoReturnUnconstrained
    module, 1377
angr.procedures.stubs.PathTerminator
    module, 1377
angr.procedures.stubs.Redirect
    module, 1377
angr.procedures.stubs.ReturnChar
    module, 1377
angr.procedures.stubs.ReturnUnconstrained
    module, 1377
angr.procedures.stubs.syscall_stub
    module, 1380
angr.procedures.stubs.UnresolvableCallTarget
    module, 1377
angr.procedures.stubs.UnresolvableJumpTarget
    module, 1377

angr.procedures.stubs.UserHook
    module, 1378
angr.procedures.testing
    module, 1380
angr.procedures.testing.manyargs
    module, 1380
angr.procedures.testing.retreg
    module, 1380
angr.procedures.tracer
    module, 1380
angr.procedures.tracer.random
    module, 1381
angr.procedures.tracer.receive
    module, 1381
angr.procedures.tracer.transmit
    module, 1381
angr.procedures.uclibc
    module, 1381
angr.procedures.win32
    module, 1381
angr.procedures.win32.critical_section
    module, 1383
angr.procedures.win32.dynamic_loading
    module, 1383
angr.procedures.win32.EncodePointer
    module, 1382
angr.procedures.win32.ExitProcess
    module, 1382
angr.procedures.win32.file_handles
    module, 1384
angr.procedures.win32.GetCommandLine
    module, 1382
angr.procedures.win32.GetCurrentProcessId
    module, 1382
angr.procedures.win32.GetCurrentThreadId
    module, 1382
angr.procedures.win32.gethostbyname
    module, 1384
angr.procedures.win32.GetLastInputInfo
    module, 1382
angr.procedures.win32.GetModuleHandle
    module, 1382
angr.procedures.win32.GetProcessAffinityMask
    module, 1383
angr.procedures.win32.heap
    module, 1384
angr.procedures.win32.InterlockedExchange
    module, 1383
angr.procedures.win32.is_bad_ptr
    module, 1384
angr.procedures.win32.IsProcessorFeaturePresent
    module, 1383
angr.procedures.win32.local_storage
    module, 1385
angr.procedures.win32.mutex
    module, 1385
angr.procedures.win32.sim_time
    module, 1386
angr.procedures.win32.system_paths
    module, 1387
angr.procedures.win32.VirtualAlloc
    module, 1383
angr.procedures.win32.VirtualProtect
    module, 1383
angr.procedures.win32_kernel
    module, 1387
angr.procedures.win32_kernel.ExAllocatePool
    module, 1387
angr.procedures.win32_kernel.ExFreePoolWithTag
    module, 1387
angr.procedures.win_user32
    module, 1387
angr.procedures.win_user32.chars
    module, 1387
angr.procedures.win_user32.keyboard
    module, 1387
angr.procedures.win_user32.messagebox
    module, 1388
angr.project
    module, 1388
angr.protos
    module, 1393
angr.protos.ail_types_pb2
    module, 1393
angr.protos.cfg_pb2
    module, 1395
angr.protos.clinic_pb2
    module, 1396
angr.protos.codegen_pb2
    module, 1397
angr.protos.decompilation_cache_pb2
    module, 1402
angr.protos.function_pb2
    module, 1402
angr.protos.primitives_pb2
    module, 1403
angr.protos.variables_pb2
    module, 1405
angr.protos.xrefs_pb2
    module, 1406
angr.rust
    module, 1406
angr.rust.analyses
    module, 1406
angr.rust.analyses.cleanup_function_identification
    module, 1406
angr.rust.analyses.flirt_sig_propagation
    module, 1407

angr.rust.analyses.rust_calling_convention
    module, 1407
angr.rust.analyses.rust_calling_convention.fact_collector
    module, 1408
angr.rust.analyses.rust_calling_convention.pattern
    module, 1409
angr.rust.analyses.rust_calling_convention.rust_conventions
    module, 1409
angr.rust.analyses.rust_calling_convention.rust_signature
    module, 1410
angr.rust.analyses.rust_symbol_recovery
    module, 1410
angr.rust.analyses.rust_type_hints
    module, 1410
angr.rust.analyses.rustc_version_identification
    module, 1410
angr.rust.analyses.struct_builder
    module, 1410
angr.rust.analyses.type_db_loader
    module, 1411
angr.rust.knowledge_plugins
    module, 1411
angr.rust.knowledge_plugins.clinic_factory
    module, 1412
angr.rust.knowledge_plugins.known_structs
    module, 1412
angr.rust.knowledge_plugins.librust
    module, 1413
angr.rust.knowledge_plugins.rust_calling_conventions
    module, 1413
angr.rust.knowledge_plugins.type_hints
    module, 1413
angr.rust.mixins
    module, 1413
angr.rust.mixins.cfa_mixin
    module, 1417
angr.rust.mixins.cfg_transformation_mixin
    module, 1418
angr.rust.mixins.dfa_mixin
    module, 1418
angr.rust.mixins.srda_mixin
    module, 1419
angr.rust.mixins.ssa_variable_mixin
    module, 1420
angr.rust.optimization_passes
    module, 1421
angr.rust.optimization_passes.cleanup_code_remover
    module, 1421
angr.rust.optimization_passes.combo_register_saving_remover
    module, 1421
angr.rust.optimization_passes.deref_coercion_simplifier
    module, 1422
angr.rust.optimization_passes.deref_coercion_simplifier_utils
    module, 1422
angr.rust.optimization_passes.error_propagation_simplifier
    module, 1423
angr.rust.optimization_passes.function_prototype_inference
    module, 1423
angr.rust.optimization_passes.macro
    module, 1423
angr.rust.optimization_passes.macro.format_macro_simplifier
    module, 1423
angr.rust.optimization_passes.outliners
    module, 1424
angr.rust.optimization_passes.outliners.string_cmp_outliner
    module, 1424
angr.rust.optimization_passes.outliners.string_literal_outliner
    module, 1425
angr.rust.optimization_passes.outliners.string_outliner
    module, 1425
angr.rust.optimization_passes.outliners.unwrap_outliner
    module, 1425
angr.rust.optimization_passes.outliners.vec_outliner
    module, 1426
angr.rust.optimization_passes.pattern_match_simplifier
    module, 1426
angr.rust.optimization_passes.pre_pattern_match_simplifier
    module, 1426
angr.rust.optimization_passes.redundant_block_remover
    module, 1428
angr.rust.optimization_passes.ret_expr_rewriter
    module, 1428
angr.rust.optimization_passes.rust_calling_convention
    module, 1428
angr.rust.optimization_passes.security_check_remover
    module, 1428
angr.rust.optimization_passes.str_argument_simplifier
    module, 1429
angr.rust.optimization_passes.struct_instantiation_simplifier
    module, 1429
angr.rust.optimization_passes.struct_return_simplifier
    module, 1430
angr.rust.optimization_passes.utils
    module, 1430
angr.rust.sim_type
    module, 1431
angr.rust.structuring
    module, 1435
angr.rust.structuring.structurer_nodes
    module, 1436
angr.rust.typehoon
    module, 1436
angr.rust.typehoon.translator
    module, 1436
angr.rust.typehoon.typehoon
    module, 1437
angr.rust.utils
    module, 1437

angr.rust.utils.ail
    module, 1437
angr.rust.utils.demangler
    module, 1438
angr.rust.utils.rust_sigs
    module, 1439
angr.rustylib
    module, 1439
angr.serializable
    module, 1440
angr.sim_manager
    module, 1441
angr.sim_options
    module, 1448
angr.sim_procedure
    module, 1448
angr.sim_state
    module, 1452
angr.sim_state_options
    module, 1456
angr.sim_type
    module, 1459
angr.sim_variable
    module, 1477
angr.simos
    module, 1479
angr.simos.cgc
    module, 1486
angr.simos.javavm
    module, 1486
angr.simos.linux
    module, 1488
angr.simos.simos
    module, 1488
angr.simos.snimmuc_nxp
    module, 1491
angr.simos.uefi
    module, 1491
angr.simos.userland
    module, 1491
angr.simos.windows
    module, 1492
angr.simos.xbox
    module, 1493
angr.slicer
    module, 1493
angr.state_hierarchy
    module, 1494
angr.state_plugins
    module, 1494
angr.state_plugins.callstack
    module, 1542
angr.state_plugins.cgc
    module, 1544
angr.state_plugins.debug_variables
    module, 1545
angr.state_plugins.edge_hitmap
    module, 1547
angr.state_plugins.filesystem
    module, 1547
angr.state_plugins.gdb
    module, 1551
angr.state_plugins.globals
    module, 1551
angr.state_plugins.heap
    module, 1552
angr.state_plugins.heap.heap_base
    module, 1555
angr.state_plugins.heap.heap_brk
    module, 1556
angr.state_plugins.heap.heap_freelist
    module, 1557
angr.state_plugins.heap.heap_libc
    module, 1558
angr.state_plugins.heap.heap_ptmalloc
    module, 1559
angr.state_plugins.heap.utils
    module, 1561
angr.state_plugins.history
    module, 1561
angr.state_plugins.icicle
    module, 1563
angr.state_plugins.inspect
    module, 1565
angr.state_plugins.javavm_classloader
    module, 1573
angr.state_plugins.jni_references
    module, 1573
angr.state_plugins.libc
    module, 1574
angr.state_plugins.light_registers
    module, 1577
angr.state_plugins.log
    module, 1577
angr.state_plugins.loop_data
    module, 1577
angr.state_plugins.plugin
    module, 1578
angr.state_plugins.posix
    module, 1579
angr.state_plugins.preconstrainer
    module, 1582
angr.state_plugins.scratch
    module, 1583
angr.state_plugins.sim_action
    module, 1584
angr.state_plugins.sim_action_object
    module, 1585

angr.state_plugins.sim_event
    module, 1587
angr.state_plugins.solver
    module, 1587
angr.state_plugins.symbolizer
    module, 1596
angr.state_plugins.trace_additions
    module, 1596
angr.state_plugins.uc_manager
    module, 1598
angr.state_plugins.unicorn_engine
    module, 1599
angr.state_plugins.view
    module, 1604
angr.storage
    module, 1607
angr.storage.file
    module, 1609
angr.storage.memory_mixins
    module, 1617
angr.storage.memory_mixins.actions_mixin
    module, 1633
angr.storage.memory_mixins.address_concretization_mixin
    module, 1633
angr.storage.memory_mixins.bvv_conversion_mixin
    module, 1634
angr.storage.memory_mixins.conditional_store_mixin
    module, 1634
angr.storage.memory_mixins.convenient_mappings_mixin
    module, 1634
angr.storage.memory_mixins.default_filler_mixin
    module, 1635
angr.storage.memory_mixins.dirty_addrs_mixin
    module, 1635
angr.storage.memory_mixins.hex_dumper_mixin
    module, 1635
angr.storage.memory_mixins.inspect_mixin
    module, 1636
angr.storage.memory_mixins.javavm_memory_mixin
    module, 1636
angr.storage.memory_mixins.keyvalue_memory_mixin
    module, 1637
angr.storage.memory_mixins.label_merger_mixin
    module, 1638
angr.storage.memory_mixins.memory_mixin
    module, 1638
angr.storage.memory_mixins.multi_value_merger_mixin
    module, 1641
angr.storage.memory_mixins.name_resolution_mixin
    module, 1641
angr.storage.memory_mixins.paged_memory
    module, 1641
angr.storage.memory_mixins.paged_memory.page_backer_mixins
    module, 1641
angr.storage.memory_mixins.paged_memory.paged_memory_mixin
    module, 1641
angr.storage.memory_mixins.paged_memory.paged_memory_multi
    module, 1643
angr.storage.memory_mixins.paged_memory.pages
    module, 1643
angr.storage.memory_mixins.paged_memory.pages.base
    module, 1647
angr.storage.memory_mixins.paged_memory.pages.cooperation
    module, 1647
angr.storage.memory_mixins.paged_memory.pages.history_trac
    module, 1648
angr.storage.memory_mixins.paged_memory.pages.ispo_mixin
    module, 1648
angr.storage.memory_mixins.paged_memory.pages.list_page
    module, 1649
angr.storage.memory_mixins.paged_memory.pages.multi_values
    module, 1649
angr.storage.memory_mixins.paged_memory.pages.mv_list_page
    module, 1651
angr.storage.memory_mixins.paged_memory.pages.permissions_
    module, 1651
angr.storage.memory_mixins.paged_memory.pages.refcount_mix
    module, 1652
angr.storage.memory_mixins.paged_memory.pages.symbolic_bit
    module, 1652
angr.storage.memory_mixins.paged_memory.pages.ultra_page
    module, 1654
angr.storage.memory_mixins.paged_memory.privileged_mixin
    module, 1655
angr.storage.memory_mixins.paged_memory.stack_allocation_m
    module, 1655
angr.storage.memory_mixins.regioned_memory
    module, 1656
angr.storage.memory_mixins.regioned_memory.abstract_addres
    module, 1659
angr.storage.memory_mixins.regioned_memory.abstract_merger
    module, 1659
angr.storage.memory_mixins.regioned_memory.region_category
    module, 1659
angr.storage.memory_mixins.regioned_memory.region_data
    module, 1659
angr.storage.memory_mixins.regioned_memory.region_meta_mix
    module, 1661
angr.storage.memory_mixins.regioned_memory.regioned_addres
    module, 1662
angr.storage.memory_mixins.regioned_memory.regioned_memory
    module, 1663
angr.storage.memory_mixins.regioned_memory.static_find_mix
    module, 1664
angr.storage.memory_mixins.simple_interface_mixin
    module, 1664
angr.storage.memory_mixins.simplification_mixin
    module, 1665

angr.storage.memory_mixins.size_resolution_mixin
    module, 1665
angr.storage.memory_mixins.slotted_memory
    module, 1665
angr.storage.memory_mixins.smart_find_mixin
    module, 1666
angr.storage.memory_mixins.symbolic_merger_mixin
    module, 1666
angr.storage.memory_mixins.top_merger_mixin
    module, 1666
angr.storage.memory_mixins.underconstrained_mixin
    module, 1666
angr.storage.memory_mixins.unwrapper_mixin
    module, 1666
angr.storage.memory_object
    module, 1666
angr.tablespecs
    module, 1668
angr.utils
    module, 1668
angr.utils.ail
    module, 1669
angr.utils.ail_serialization
    module, 1671
angr.utils.algo
    module, 1673
angr.utils.balancer
    module, 1673
angr.utils.bits
    module, 1674
angr.utils.constants
    module, 1675
angr.utils.cowdict
    module, 1676
angr.utils.cpp
    module, 1677
angr.utils.doms
    module, 1677
angr.utils.dynamic_dictlist
    module, 1678
angr.utils.endness
    module, 1678
angr.utils.enums_conv
    module, 1679
angr.utils.env
    module, 1679
angr.utils.formatting
    module, 1679
angr.utils.funcid
    module, 1680
angr.utils.go_runtime
    module, 1680
angr.utils.graph
    module, 1682
angr.utils.hashing
    module, 1689
angr.utils.ins_addr_list
    module, 1689
angr.utils.json_utils
    module, 1690
angr.utils.lazy_import
    module, 1691
angr.utils.library
    module, 1691
angr.utils.loader
    module, 1693
angr.utils.mp
    module, 1693
angr.utils.orderedset
    module, 1695
angr.utils.smart_cache
    module, 1695
angr.utils.ssa
    module, 1695
angr.utils.ssa.combined_uses_collector
    module, 1698
angr.utils.ssa.tmp_uses_collector
    module, 1699
angr.utils.ssa.vvar_extra_defs_collector
    module, 1699
angr.utils.ssa.vvar_uses_collector
    module, 1699
angr.utils.strings
    module, 1699
angr.utils.tagged_interval_map
    module, 1700
angr.utils.timing
    module, 1700
angr.utils.types
    module, 1701
angr.utils.vex
    module, 1703
angr.vaults
    module, 1704
AngrAIError, 1087
AngrAnalysisError, 1081
AngrAnnotatedCFGError, 1081
AngrAssemblyError, 1082
AngrBackwardSlicingError, 1081
AngrBladeError, 1081
AngrBladeSimProcError, 1081
AngrCallableError, 1082
AngrCallableMultistateError, 1082
AngrCFGError, 1082
AngrCorruptDBError, 1083
AngrDataGraphError, 1082
AngrDB (*class in angr.angrdb.db*), 940
AngrDBError, 1083

AngrDDGError, 1082
AngrDecompilationComplexityError, 1086
AngrDecompilationError, 1086
AngrDelayJobNotice, 1082
AngrDirectorError, 1083
AngrError, 1081
AngrExitError, 1081
AngrExplorationTechniqueError, 1083
AngrExplorerError, 1083
AngrForwardAnalysisError, 1082
AngrIncompatibleDBError, 1083
AngrIncongruencyError, 1082
AngrInvalidArgumentError, 1081
AngrJobMergingFailureNotice, 1082
AngrJobWideningFailureNotice, 1082
AngrLifterError, 1081
AngrLoopAnalysisError, 1082
AngrMissingTypeError, 1082
AngrNoPluginError, 1086
AngrObjectFactory (*class in angr.factory*), 1109
AngrPathError, 1081
AngrRuntimeDbError, 1083
AngrRuntimeError, 1081
AngrSimOSError, 1082
AngrSkipJobNotice, 1082
AngrSurveyorError, 1081
AngrSyscallError, 1082
AngrTracerError, 1083
AngrTypeError, 1082
AngrUnsupportedSyscallError, 1085
AngrValueError, 1081
AngrVariableRecoveryError, 1083
AngrVaultError, 1081
AngrVFGError, 1082
AngrVFGRestartAnalysisNotice, 1082
annotate_mv_with_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 861
annotate_with_def() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 861
annotate_with_def() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1251
annotate_with_variables() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 928
annotated_cfg() (*angr.analyses.backward_slice.BackwardSlice method*), 278
AnnotatedCFG (*class in angr.annocfg*), 957
annotations (*angr.state_plugins.sim_action_object.SimActionObject property*), 1586
ansi_color() (*in module angr.utils.formatting*), 1679
any_of() (*in module angr.analyses.decompiler.known_patterns.gating*), 515
any_set() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_b method*), 1654
AnyOf (*class in angr.analyses.decompiler.known_patterns.gating*), 513
APIDeobFuncDescriptor (*class in angr.analyses.deobfuscator.api_obf_finder*), 734
APIObfType1PeepholeOptimizer (*class in angr.analyses.deobfuscator.api_obf_peephole_optimizer*), 735
APIObfType3PeepholeOptimizer (*class in angr.analyses.deobfuscator.api_obf_peephole_optimizer*), 735
APIObfuscationFinder (*class in angr.analyses.deobfuscator.api_obf_finder*), 735
APIObfuscationType (*class in angr.analyses.deobfuscator.api_obf_finder*), 734
APIObfuscationType2 (*class in angr.analyses.deobfuscator.api_obf_type2_finder*), 736
APIObfuscationType2Finder (*class in angr.analyses.deobfuscator.api_obf_type2_finder*), 736
APP_16_BIT (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_32_BIT (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_64_BIT (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_CONSOLE (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_DLL (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_DRV (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_EXE (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_GRAPHICS (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_MULTI_THREADED (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
APP_SINGLE_THREADED (*angr.analyses.flirt.consts.FlirtAppType attribute*), 754
append_args() (*angr.tablespecs.StringTableSpec method*), 1668
append_data() (*angr.analyses.reassembler.Reassembler method*), 877
append_env() (*angr.tablespecs.StringTableSpec method*), 1668

append_procedure() (*angr.analyses.reassembler.Reassembler method*), 877
append_state() (*angr.analyses.vfg.VFGNode method*), 934
append_statement() (*angr.analyses.decompiler.dephication.rewriting_engine.SimEngineDephiRewriting method*), 420
append_statement() (*angr.analyses.decompiler.ssailification.rewriting_engine.RewritingState method*), 655
append_statement() (*in module angr.analyses.decompiler.utils*), 720
applicable() (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern method*), 521
applicable() (*angr.analyses.decompiler.known_patterns.templates.KnownPatternTemplate method*), 532
apply() (*angr.analyses.cfg.cfg_fast.FunctionCallEdge method*), 319
apply() (*angr.analyses.cfg.cfg_fast.FunctionEdge method*), 318
apply() (*angr.analyses.cfg.cfg_fast.FunctionFakeRetEdge method*), 319
apply() (*angr.analyses.cfg.cfg_fast.FunctionReturnEdge method*), 319
apply() (*angr.analyses.cfg.cfg_fast.FunctionTransitionEdge method*), 318
apply() (*angr.calling_conventions.AllocHelper method*), 963
apply() (*angr.sim_manager.SimulationManager method*), 1446
apply_at_callsite (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 839
apply_call_info_to_graph() (*in module angr.analyses.decompiler.known_patterns.apply*), 485
apply_definition() (*angr.knowledge_plugins.functions.function.Function method*), 1217
apply_function_edges() (*angr.analyses.cfg.cfg_fast.CFGJob method*), 320
apply_matches() (*angr.analyses.decompiler.known_patterns.finder.KnownPatternFinder method*), 509
apply_names() (*angr.analyses.decompiler.semantic_naming.naming_base.SemanticNamingBase method*), 649
apply_patches_to_binary() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
apply_patches_to_state() (*angr.knowledge_plugins.patches.PatchManager method*), 1266
arch (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions attribute*), 307
arch (*angr.analyses.cfg.CFGArchOptions attribute*), 292
arch (*angr.analyses.CFGArchOptions attribute*), 238
ARCH (*angr.calling_conventions.SimCC attribute*), 967
arch (*angr.sim_state.SimState property*), 1454
ARCH (*angr.SimCC attribute*), 173
ARCH() (*angr.engines.pcode.cc.SimCCPCodeBase class method*), 1034
ARCH_386 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_6502 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_6800 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_68K (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_80196 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_8051 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_AD218X (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_ALPHA (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_ARM (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_AVR (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_C166 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_C39 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_CR16 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_DALVIK (*angr.analyses.flirt.consts.FlirtArch attribute*), 753
ARCH_DSP56K (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_EBC (*angr.analyses.flirt.consts.FlirtArch attribute*), 753
ARCH_F2MC (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_FR (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_H8 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_H8500 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_HPPA (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_I860 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_I960 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_IA64 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_JAVA (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_KR1878 (*angr.analyses.flirt.consts.FlirtArch*

*attribute*), 752
ARCH_M32R (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_M740 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_M7700 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_M7900 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_MC6812 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_MC6816 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_MIPS (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_MN102L00 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_MSP430 (*angr.analyses.flirt.consts.FlirtArch attribute*), 753
ARCH_NEC_78K0 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_NEC_78K0S (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_NEC_V850X (*angr.analyses.flirt.consts.FlirtArch attribute*), 753
ARCH_NET (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_OAKDSP (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
arch_overridable() (*in module angr.sim_state*), 1452
ARCH_PDP (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_PIC (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_PPC (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
arch_prctl (*class in angr.procedures.linux_kernel.arch_prctl*), 1364
ARCH_SCR_ADPT (*angr.analyses.flirt.consts.FlirtArch attribute*), 753
ARCH_SH (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_SPARC (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_SPU (*angr.analyses.flirt.consts.FlirtArch attribute*), 753
ARCH_ST20 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_ST7 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_ST9 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_TLCS900 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_TMS (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_TMS320C1X (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_TMS320C3 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_TMS320C54 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_TMS320C55 (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_TMSC6 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_TRICORE (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_TRIMEDIA (*angr.analyses.flirt.consts.FlirtArch attribute*), 752
ARCH_Z8 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
ARCH_Z80 (*angr.analyses.flirt.consts.FlirtArch attribute*), 751
arches (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 457
arches (*angr.analyses.decompiler.known_patterns.KnownPatternTemplate attribute*), 463
arches (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521
arches (*angr.analyses.decompiler.known_patterns.templates.KnownPattern attribute*), 532
arches (*angr.analyses.decompiler.KnownPattern attribute*), 379
ARCHES (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simp attribute*), 560
ARCHES (*angr.analyses.decompiler.optimization_passes.BasePointerSaveSim attribute*), 540
ARCHES (*angr.analyses.decompiler.optimization_passes.call_stmt_rewriter.C attribute*), 560
ARCHES (*angr.analyses.decompiler.optimization_passes.CallStatementRewri attribute*), 540
ARCHES (*angr.analyses.decompiler.optimization_passes.code_motion.CodeM attribute*), 561
ARCHES (*angr.analyses.decompiler.optimization_passes.CodeMotionOptimi attribute*), 541
ARCHES (*angr.analyses.decompiler.optimization_passes.condition_constpro attribute*), 562
ARCHES (*angr.analyses.decompiler.optimization_passes.ConditionConstantP attribute*), 542
ARCHES (*angr.analyses.decompiler.optimization_passes.const_derefs.Consta attribute*), 563
ARCHES (*angr.analyses.decompiler.optimization_passes.const_prop_reverter attribute*), 563
ARCHES (*angr.analyses.decompiler.optimization_passes.ConstantDereferenc attribute*), 543
ARCHES (*angr.analyses.decompiler.optimization_passes.ConstPropOptRever attribute*), 542

ARCHES (*angr.analyses.decompiler.optimization_passes.dead_assignments.DeadAssignments attribute*), 565
ARCHES (*angr.analyses.decompiler.optimization_passes.DeadAssignments attribute*), 544
ARCHES (*angr.analyses.decompiler.optimization_passes.determine_loop_initializers.DetermineLoopInitializers attribute*), 565
ARCHES (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimplifier attribute*), 566
ARCHES (*angr.analyses.decompiler.optimization_passes.DivSimplifier attribute*), 544
ARCHES (*angr.analyses.decompiler.optimization_passes.eager_returns.EagerReturnsSimplifier attribute*), 573
ARCHES (*angr.analyses.decompiler.optimization_passes.eager_returns.EagerReturnsSimplifierLoopPass attribute*), 574
ARCHES (*angr.analyses.decompiler.optimization_passes.EagerReturnsSimplifier attribute*), 546
ARCHES (*angr.analyses.decompiler.optimization_passes.expression_narrower.ExpressionNarrower attribute*), 576
ARCHES (*angr.analyses.decompiler.optimization_passes.ExpressionNarrower attribute*), 546
ARCHES (*angr.analyses.decompiler.optimization_passes.flip_boolean_cmp.FlipBooleanCmp attribute*), 576
ARCHES (*angr.analyses.decompiler.optimization_passes.FlipBooleanCmp attribute*), 547
ARCHES (*angr.analyses.decompiler.optimization_passes.inline_small_functions.InlineSmallFunctions attribute*), 577
ARCHES (*angr.analyses.decompiler.optimization_passes.inline_wrappers.InlineWrappers attribute*), 578
ARCHES (*angr.analyses.decompiler.optimization_passes.inlined_strcpy_consolidation.InlinedStrcpyConsolidation attribute*), 578
ARCHES (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 581
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedStrcpyConsolidation attribute*), 581
ARCHES (*angr.analyses.decompiler.optimization_passes.InlinedStringTransformationSimplifier attribute*), 581
ARCHES (*angr.analyses.decompiler.optimization_passes.InlineSmallFunctions attribute*), 548
ARCHES (*angr.analyses.decompiler.optimization_passes.InlineWrappers attribute*), 548
ARCHES (*angr.analyses.decompiler.optimization_passes.ITEExprConverter attribute*), 549
ARCHES (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 549
ARCHES (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 582
ARCHES (*angr.analyses.decompiler.optimization_passes.ite_region_converter.ITERegionConverter attribute*), 583
ARCHES (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 547
ARCHES (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 547
ARCHES (*angr.analyses.decompiler.optimization_passes.known_pattern_outliner.KnownPatternOutliner attribute*), 583
ARCHES (*angr.analyses.decompiler.optimization_passes.KnownPatternOutliner attribute*), 550
ARCHES (*angr.analyses.decompiler.optimization_passes.mips_gp_setting_simplifier.MipsGpSettingSimplifier attribute*), 585
ARCHES (*angr.analyses.decompiler.optimization_passes.MipsGpSettingSimplifier attribute*), 551
ARCHES (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 586
ARCHES (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 552
ARCHES (*angr.analyses.decompiler.optimization_passes.optimization_pass.BaseOptimizationPass attribute*), 587
ARCHES (*angr.analyses.decompiler.optimization_passes.optimization_pass.StructuringOptimizationPass attribute*), 589
ARCHES (*angr.analyses.decompiler.optimization_passes.peephole_simplifier_multi_block.PeepholeSimplifierMultiBlock attribute*), 590
ARCHES (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 590
ARCHES (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier_adv.RegisterSaveAreaSimplifierAdvanced attribute*), 553
ARCHES (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifier attribute*), 553
ARCHES (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 591
ARCHES (*angr.analyses.decompiler.optimization_passes.return_duplication_base.ReturnDuplicationBase attribute*), 591
ARCHES (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSimplifier attribute*), 553
ARCHES (*angr.analyses.decompiler.optimization_passes.return_duplicator_base.CodeCopy attribute*), 592
ARCHES (*angr.analyses.decompiler.optimization_passes.return_duplicator_base.CodeNode attribute*), 593
ARCHES (*angr.analyses.decompiler.optimization_passes.ReturnDeduplicator attribute*), 554
ARCHES (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHighLevel attribute*), 554
ARCHES (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorLowLevel attribute*), 555
ARCHES (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 594
ARCHES (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifier attribute*), 556
ARCHES (*angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.StaticVVarRewriter attribute*), 596
ARCHES (*angr.analyses.decompiler.optimization_passes.StaticVVarRewriter attribute*), 556
ARCHES (*angr.analyses.decompiler.optimization_passes.switch_default_case_simplifier.SwitchDefaultCaseSimplifier attribute*), 596

ARCHES (*angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator.SwitchDefaultCaseDuplicator attribute*), 597
ARCHES (*angr.analyses.decompiler.optimization_passes.SwitchExpressionSimplifier attribute*), 556
ARCHES (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryRewriter attribute*), 557
ARCHES (*angr.analyses.decompiler.optimization_passes.tag_based_cascading_if_converter.TagBasedCascadingIfConverter attribute*), 597
ARCHES (*angr.analyses.decompiler.optimization_passes.TagSkeleton attribute*), 558
ARCHES (*angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier.WinStackCanarySimplifier attribute*), 598
ARCHES (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier attribute*), 558
ARCHES (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 598
ARCHES (*angr.analyses.decompiler.optimization_passes.X86GccGetPcSimplifier attribute*), 558
ARCHES (*angr.analyses.deobfuscator.string_obf_opt_passes.StringObfOptPassRewriter attribute*), 739
ARCHES (*angr.analyses.deobfuscator.StringObfType3Rewriter attribute*), 733
ARCHES (*angr.rust.optimization_passes.cleanup_code_remover.CleanupCodeRemover attribute*), 1421
ARCHES (*angr.rust.optimization_passes.combo_register_rewriter.ComboRegisterRewriter attribute*), 1421
ARCHES (*angr.rust.optimization_passes.deref_coercion_simplifier.DerefCoercionSimplifier attribute*), 1422
ARCHES (*angr.rust.optimization_passes.deref_coercion_simplifier.DerefCoercionSimplifier attribute*), 1422
ARCHES (*angr.rust.optimization_passes.error_propagation_simplifier.ErrorPropagationSimplifier attribute*), 1423
ARCHES (*angr.rust.optimization_passes.function_prototype_inference.FunctionPrototypeInference attribute*), 1423
ARCHES (*angr.rust.optimization_passes.macro.format_macro_simplifier.FormatMacroSimplifier attribute*), 1424
ARCHES (*angr.rust.optimization_passes.outliners.string_cmp_outliner.StringCmpOutliner attribute*), 1424
ARCHES (*angr.rust.optimization_passes.outliners.string_literal_outliner.StringLiteralOutliner attribute*), 1425
ARCHES (*angr.rust.optimization_passes.outliners.string_outliner.StringOutliner attribute*), 1425
ARCHES (*angr.rust.optimization_passes.outliners.unwrap_outliner.UnwrapOutliner attribute*), 1425
ARCHES (*angr.rust.optimization_passes.outliners.vec_outliner.VecOutliner attribute*), 1426
ARCHES (*angr.rust.optimization_passes.pattern_match_simplifier.PatternMatchSimplifier attribute*), 1426
ARCHES (*angr.rust.optimization_passes.pre_pattern_match_simplifier.PrePatternMatchSimplifier attribute*), 1427
ARCHES (*angr.rust.optimization_passes.redundant_block_remover.RedundantBlockRemover attribute*), 1428
ARCHES (*angr.rust.optimization_passes.ret_expr_rewriter.RetExprRewriter attribute*), 1428
ARCHES (*angr.rust.optimization_passes.security_check_remover.SecurityCheckRemover attribute*), 1428
ARCHES (*angr.rust.optimization_passes.str_argument_simplifier.StrArgumentSimplifier attribute*), 1429
ARCHES (*angr.rust.optimization_passes.struct_instantiation_simplifier.StructInstantiationSimplifier attribute*), 1430
ARCHES (*angr.rust.optimization_passes.struct_return_simplifier.StructReturnSimplifier attribute*), 1430
arches() (*angr.calling_conventions.SimCC class method*), 971
ArgExpr (*angr.analyses.decompiler.edits.resolve.ResolvedVariable attribute*), 447
arg_index (*angr.analyses.decompiler.edits.resolve.ResolvedVariable attribute*), 428
arg_locs() (*angr.calling_conventions.SimCC method*), 969
arg_locs() (*angr.calling_conventions.SimCCMicrosoftThiscall method*), 972
arg_reg_offsets() (*angr.calling_conventions.SimCC class method*), 967
ARG_REG_SANITY_FILTER (*angr.calling_conventions.SimCC attribute*), 968
ARG_REG_SANITY_FILTER (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
ARG_REG_SANITY_FILTER (*angr.SimCC attribute*), 173
ARG_REGS (*angr.calling_conventions.SimCC attribute*), 967
ARG_REGS (*angr.calling_conventions.SimCCAArch64 attribute*), 978
ARG_REGS (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 978
ARG_REGS (*angr.calling_conventions.SimCCAArch64WindowsSyscall attribute*), 978
ARG_REGS (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 976
ARG_REGS (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 976
ARG_REGS (*angr.calling_conventions.SimCCARM attribute*), 976
ARG_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 977
ARG_REGS (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 977
ARG_REGS (*angr.calling_conventions.SimCCARMWindowsSyscall attribute*), 978
ARG_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 974
ARG_REGS (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
ARG_REGS (*angr.calling_conventions.SimCCGoAMD64ABI0*

*attribute*), 976
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 972
ARG_REGS (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 972
ARG_REGS (*angr.calling_conventions.SimCCN64 attribute*), 980
ARG_REGS (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 981
ARG_REGS (*angr.calling_conventions.SimCCO32 attribute*), 979
ARG_REGS (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 980
ARG_REGS (*angr.calling_conventions.SimCCPowerPC attribute*), 981
ARG_REGS (*angr.calling_conventions.SimCCPowerPC64 attribute*), 982
ARG_REGS (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 982
ARG_REGS (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 981
ARG_REGS (*angr.calling_conventions.SimCCRISCV64 attribute*), 979
ARG_REGS (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 979
ARG_REGS (*angr.calling_conventions.SimCCS390X attribute*), 982
ARG_REGS (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 983
ARG_REGS (*angr.calling_conventions.SimCCSoot attribute*), 982
ARG_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
ARG_REGS (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 974
ARG_REGS (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 974
ARG_REGS (*angr.engines.pcode.cc.SimCCM68k attribute*), 1034
ARG_REGS (*angr.engines.pcode.cc.SimCCPARISC attribute*), 1035
ARG_REGS (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 1035
ARG_REGS (*angr.engines.pcode.cc.SimCCRISCV attribute*), 1035
ARG_REGS (*angr.engines.pcode.cc.SimCCSH4 attribute*), 1035
ARG_REGS (*angr.engines.pcode.cc.SimCCSPARC attribute*), 1035
ARG_REGS (*angr.engines.pcode.cc.SimCCXtensa attribute*), 1036
ARG_REGS (*angr.SimCC attribute*), 173
arg_session() (*angr.calling_conventions.SimCC method*), 968
ARG_SLOT_SIZE (*angr.calling_conventions.SimCC attribute*), 967
arg_slot_size (*angr.calling_conventions.SimCC property*), 968
ARG_SLOT_SIZE (*angr.calling_conventions.SimCCN32 attribute*), 980
ARG_SLOT_SIZE (*angr.calling_conventions.SimCCN32LinuxSyscall attribute*), 981
ARG_SLOT_SIZE (*angr.SimCC attribute*), 173
args (*angr.analyses.decompiler.known_patterns.dsl.PCall attribute*), 497
args (*angr.analyses.decompiler.known_patterns.PCall attribute*), 466
args (*angr.state_plugins.sim_action_object.SimActionObject property*), 1586
args() (*angr.analyses.identifier.functions.based_atoi.based_atoi method*), 772
args() (*angr.analyses.identifier.functions.fdprintf.fdprintf method*), 773
args() (*angr.analyses.identifier.functions.int2str.int2str method*), 774
args() (*angr.analyses.identifier.functions.int2str.int2str_v2 method*), 774
args() (*angr.analyses.identifier.functions.int2str.int2str_v3 method*), 774
args() (*angr.analyses.identifier.functions.int2str.int2str_v4 method*), 774
args() (*angr.analyses.identifier.functions.memcmp.memcmp method*), 775
args() (*angr.analyses.identifier.functions.memcpy.memcpy method*), 775
args() (*angr.analyses.identifier.functions.memset.memset method*), 776
args() (*angr.analyses.identifier.functions.printf.printf method*), 777
args() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 778
args() (*angr.analyses.identifier.functions.recv_until.receive_until_fd method*), 777
args() (*angr.analyses.identifier.functions.skip_calloc.calloc method*), 778
args() (*angr.analyses.identifier.functions.skip_realloc.realloc method*), 778
args() (*angr.analyses.identifier.functions.skip_recv_n.receive_n2 method*), 779
args() (*angr.analyses.identifier.functions.skip_recv_n.receive_n3 method*), 779
args() (*angr.analyses.identifier.functions.skip_recv_n.receive_n4 method*), 778
args() (*angr.analyses.identifier.functions.snprintf.snprintf method*), 779
args() (*angr.analyses.identifier.functions.sprintf.sprintf*

*method*), 780
args() (*angr.analyses.identifier.functions.strcmp.strcmp method*), 782
args() (*angr.analyses.identifier.functions.strcpy.strcpy method*), 782
args() (*angr.analyses.identifier.functions.strncmp.strncmp method*), 783
args() (*angr.analyses.identifier.functions.strncpy.strncpy method*), 784
args() (*angr.analyses.identifier.functions.strtol.strtol method*), 784
args_atoms (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
args_atoms (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
ARGS_MISMATCH (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1379
ARGS_MISMATCH (*angr.procedures.stubs.ReturnUnconstrained.ReturnUnconstrained attribute*), 1377
ARGS_MISMATCH (*angr.sim_procedure.SimProcedure attribute*), 1450
ARGS_MISMATCH (*angr.SimProcedure attribute*), 190
args_values (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
args_values (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
ArgSession (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
ArgSession (*angr.calling_conventions.SimCCUsercall attribute*), 971
ArgSession (*class in angr.calling_conventions*), 966
ARGUMENT (*angr.analyses.decompiler.stack_item.StackItemType attribute*), 659
argument_types (*angr.sim_procedure.SimProcedure property*), 1452
arguments (*angr.knowledge_plugins.functions.function.Function property*), 1216
ArgVVar (*class in angr.protos.ail_types_pb2*), 1393
ArgVVars (*class in angr.protos.ail_types_pb2*), 1394
ArgVVars.EntriesEntry (*class in angr.protos.ail_types_pb2*), 1394
ArithmeticExpression (*class in angr.engines.light.data*), 1009
ARM (*angr.analyses.cfg.cfg_fast.ARMDecodingMode attribute*), 315
arm64_zerobit() (*in module angr.engines.vex.claripy.ccall*), 1072
arm64g_32bit_truncate_operands() (*in module angr.engines.vex.claripy.ccall*), 1073
arm64g_calculate_condition() (*in module angr.engines.vex.claripy.ccall*), 1073
arm64g_calculate_flag_c() (*in module angr.engines.vex.claripy.ccall*), 1073
arm64g_calculate_flag_n() (*in module angr.engines.vex.claripy.ccall*), 1072
arm64g_calculate_flag_v() (*in module angr.engines.vex.claripy.ccall*), 1073
arm64g_calculate_flag_z() (*in module angr.engines.vex.claripy.ccall*), 1073
arm64g_calculate_flags_nzcv() (*in module angr.engines.vex.claripy.ccall*), 1073
arm64g_dirtyhelper_MRS_DCZID_EL0() (*in module angr.engines.vex.heavy.dirty*), 1078
arm_zerobit() (*in module angr.engines.vex.claripy.ccall*), 1072
ARMCallRewriteData (*class in angr.analyses.decompiler.ccall_rewriters.arm_ccalls*), 402
ARMCmpF (*class in angr.analyses.decompiler.peephole_optimizations.arm_cc*), 602
ARMDecodingMode (*class in angr.analyses.cfg.cfg_fast*), 315
ArmElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast*), 332
armg_calculate_condition() (*in module angr.engines.vex.claripy.ccall*), 1072
armg_calculate_flag_c() (*in module angr.engines.vex.claripy.ccall*), 1072
armg_calculate_flag_n() (*in module angr.engines.vex.claripy.ccall*), 1072
armg_calculate_flag_v() (*in module angr.engines.vex.claripy.ccall*), 1072
armg_calculate_flag_z() (*in module angr.engines.vex.claripy.ccall*), 1072
armg_calculate_flags_nzcv() (*in module angr.engines.vex.claripy.ccall*), 1072
Array (*class in angr.ailment.expression*), 218
Array (*class in angr.analyses.typehoon.typeconsts*), 910
array() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 1546
array() (*angr.state_plugins.view.SimMemView method*), 1607
ArrayIndexNaming (*class in angr.analyses.decompiler.semantic_naming.array_index_naming*), 647
as_address (*angr.analyses.decompiler.known_patterns.dsl.PStackField attribute*), 500
as_address (*angr.analyses.decompiler.known_patterns.PStackField attribute*), 474
as_struct_ty() (*angr.rust.sim_type.EnumVariant method*), 1434
as_struct_ty() (*angr.rust.sim_type.RustSimEnum method*), 1435
AShlConstSubA (*class in angr.analyses.decompiler.peephole_optimizations.a_shl_const_su*), 601
assembly() (*angr.analyses.reassembler.BasicBlock*

*method*), 873
assembly() (*angr.analyses.reassembler.Data method*), 874
assembly() (*angr.analyses.reassembler.Instruction method*), 872
assembly() (*angr.analyses.reassembler.Operand method*), 872
assembly() (*angr.analyses.reassembler.Procedure method*), 873
assembly() (*angr.analyses.reassembler.Reassembler method*), 877
assign() (*angr.state_plugins.uc_manager.SimUCManager method*), 1598
assign_labels() (*angr.analyses.reassembler.BasicBlock method*), 873
assign_labels() (*angr.analyses.reassembler.Data method*), 874
assign_labels() (*angr.analyses.reassembler.Instruction method*), 872
assign_labels() (*angr.analyses.reassembler.Procedure method*), 873
assign_unified_variable_names() (*angr.knowledge_plugins.variables.variable_manager.Variable*
*method*), 1287
assign_variable_names() (*angr.knowledge_plugins.variables.variable_manager.Variable*
*method*), 1287
Assignment (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.Address*
*attribute*), 334
Assignment (*class in angr.ailment.statement*), 224
AssignmentNode (*class in angr.analyses.loop_analysis.loop_analysis*), 788
AssignmentToASTVisitor (*class in angr.analyses.loop_analysis.loop_analysis*), 788
ast (*angr.analyses.data_dep.dep_nodes.BaseDepNode property*), 359
AST (*class in angr.analyses.ddg*), 361
ast_graph (*angr.analyses.ddg.DDG property*), 365
ast_preserving_op() (*in module angr.state_plugins.sim_action_object*), 1585
ast_stripping_decorator() (*in module angr.state_plugins.sim_action_object*), 1585
ast_weight() (*in module angr.exploration_techniques.suggestions*), 1106
ASTNodeBase (*class in angr.analyses.loop_analysis.loop_analysis*), 787
AstToPosEntry (*class in angr.protos.codegen_pb2*), 1397
ASubADiv (*class in angr.analyses.decompiler.peephole_optimizations.a_sub_a_div*), 601
ASubAShrConstShrConst (*class in angr.analyses.decompiler.peephole_optimizations.a_sub_a_shr_c*
601
ASubASubN (*class in angr.analyses.decompiler.peephole_optimizations.a_su*
601
at_new_block() (*angr.knowledge_plugins.key_definitions.liveness.Livenes*
*method*), 1258
at_new_block() (*angr.knowledge_plugins.key_definitions.rd_model.Reach*
*method*), 1259
at_new_stmt() (*angr.knowledge_plugins.key_definitions.liveness.Liveness*
*method*), 1258
at_new_stmt() (*angr.knowledge_plugins.key_definitions.rd_model.Reachi*
*method*), 1259
atoi (*class in angr.analyses.identifier.functions.atoi*), 772
atoi (*class in angr.procedures.libc.atoi*), 1353
atoi_dumps() (*angr.state_plugins.trace_additions.ChallRespInfo static method*), 1597
atol (*class in angr.procedures.libc.atol*), 1353
Atom (*class in angr.ailment.expression*), 218
Atom (*class in angr.knowledge_plugins.key_definitions.atoms*), 1241
atom_ctr (*angr.ailment.Manager attribute*), 212
atomCtr (*angr.ailment.manager.Manager attribute*), 224
AtomKind (*class in angr.knowledge_plugins.key_definitions.atoms*), 1241
AtypicallyAlignedFunction (*class in angr.analyses.patchfinder*), 793
auto_import_modules() (*in module angr.misc.autoimport*), 1314
auto_import_packages() (*in module angr.misc.autoimport*), 1314
auto_import_source_files() (*in module angr.misc.autoimport*), 1315
available_flavors() (*angr.knowledge_plugins.structured_code.StructuredCodeManage*
*method*), 1279

## B

b() (*angr.state_plugins.inspect.SimInspector method*), 1572
b64_decode (*class in angr.procedures.stubs.b64_decode*), 1378
back_edges() (*angr.analyses.forward_analysis.visitors.graph.GraphVisito*
*method*), 767
backend2name (*angr.angrdb.serializers.loader.LoaderSerializer attribute*), 953
backend2name (*angr.angrdb.serializers.LoaderSerializer attribute*), 948
BackgroundColor (*class in angr.misc.ansi*), 1314
backpatch() (*angr.analyses.typehoon.translator.TypeTranslator method*), 906
backtrace (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567

backtrace (*angr.state_plugins.InspectAttrs attribute*), 1498
BackwardSlice (*class in angr.analyses.backward_slice*), 277
BadJumpkindNotification, 958
BadStatesDropper (*class in angr.distributed.worker*), 994
Balancer (*class in angr.utils.balancer*), 1673
BalancerError, 1673
BalancerUnsatError, 1673
banned (*angr.analyses.pathfinder.SimStateMarker attribute*), 794
base (*angr.rust.sim_type.RustSimTypeFunction attribute*), 1432
base (*angr.sim_type.SimType attribute*), 1459
base (*angr.sim_type.SimTypeFunction attribute*), 1469
base_addr (*angr.utils.ins_addr_list.InsAddrList property*), 1690
base_addr_available (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTargetBaseAddr property*), 335
base_aliases (*angr.analyses.decompiler.known_patterns.dsl.MatchState attribute*), 490
base_aliases (*angr.analyses.decompiler.known_patterns.finder.KnownPattern attribute*), 507
base_aliases (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 461
base_aliases (*angr.analyses.decompiler.KnownPatternMatch attribute*), 383
base_args() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 778
base_args() (*angr.analyses.identifier.functions.recv_until.receive_until_fd method*), 777
base_type (*angr.sim_type.SimTypeBitfield property*), 1473
base_type (*angr.sim_type.SimTypeEnum property*), 1473
based_atoi (*class in angr.analyses.identifier.functions.based_atoi*), 772
BaseDepNode (*class in angr.analyses.data_dep.dep_nodes*), 359
BaseGoal (*class in angr.exploration_techniques.director*), 1099
BaseLabel (*class in angr.analyses.typehoon.typevars*), 919
BaseNode (*class in angr.analyses.decompiler.structurer_nodes*), 710
BaseOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 587
BasePointerOffset (*class in angr.ailment.expression*), 218
BasePointerOffsetAddN (*class in angr.analyses.decompiler.peephole_optimizations.basepointeroffs*), 605
BasePointerOffsetAndMask (*class in angr.analyses.decompiler.peephole_optimizations.basepointeroffs*), 605
BasePointerSaveSimplifier (*class in angr.analyses.decompiler.optimization_passes.base_ptr_save_sim*), 559
BaseProxiNode (*class in angr.analyses.proximity_graph*), 799
BaseStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.base*), 669
basic_spec (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1379
basic_spec (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 1380
BasicBlock (*class in angr.analyses.reassembler*), 872
basicblock_key (*angr.storage.memory_mixins.regioned_memory.region_ property*), 1662
BasicClaripyCooperation (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 1648
bbl_addr (*angr.code_location.AILCodeLocation property*), 986
bbl_addr (*angr.errors.SimError attribute*), 1083
bbl_addr (*angr.knowledge_plugins.key_definitions.definition.DefinitionMa attribute*), 1245
bbl_addr (*angr.SimError attribute*), 177
bbl_addrs (*angr.state_plugins.history.SimStateHistory property*), 1562
bck_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
bck_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 1560
BEFORE (*angr.state_plugins.inspect.When attribute*), 1570
before_comment_changed() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 440
before_comment_changed() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 442
before_func_arg_renamed() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 438
before_func_arg_renamed() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 440
before_func_arg_retyped() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 439
before_func_arg_retyped() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 441
before_function_renamed()

(*angr.analyses.decompiler.edits.hooks.EditHooks method*), 438
before_function_renamed() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 440
before_function_retyped() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 440
before_function_retyped() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 442
before_global_var_renamed() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 439
before_global_var_renamed() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 441
before_global_var_retyped() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 439
before_global_var_retyped() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 441
before_other_var_retyped() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 439
before_other_var_retyped() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 442
BEFORE_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 587
BEFORE_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 552
BEFORE_SSA_LEVEL0_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 587
BEFORE_SSA_LEVEL0_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 552
BEFORE_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 587
BEFORE_SSA_LEVEL1_TRANSFORMATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 552
before_stack_var_renamed() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 438
before_stack_var_renamed() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 440
before_stack_var_retyped() (*angr.analyses.decompiler.edits.hooks.EditHooks method*), 439
before_stack_var_retyped() (*angr.analyses.decompiler.edits.hooks.NullEditHooks method*), 441
BEFORE_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.optimization_pass attribute*), 587
BEFORE_VARIABLE_RECOVERY (*angr.analyses.decompiler.optimization_passes.OptimizationPass attribute*), 552
begin_txn() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method*), 1278
BehaviorFactory (*class in angr.engines.pcode.behavior*), 1034
bfs_list_blocks() (*in module angr.analyses.decompiler.optimization_passes.duplication_revert* 573
bfs_nodes() (*angr.analyses.decompiler.optimization_passes.optimization_* *method*), 588
binary (*angr.knowledge_plugins.functions.function.Function property*), 1213
binary_guard (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 457
binary_guard (*angr.analyses.decompiler.known_patterns.pattern.KnownP attribute*), 521
binary_guard (*angr.analyses.decompiler.KnownPattern attribute*), 379
binary_insert() (*in module angr.utils.algo*), 1673
BinaryError, 870
BinaryOp (*class in angr.analysis.expression*), 218
BinaryOptimizer (*class in angr.analyses.binary_optimizer*), 279
bind (*class in angr.procedures.posix.bind*), 1370
bind() (*angr.analyses.decompiler.known_patterns.dsl.MatchState method*), 490
BindDiff (*class in angr.analyses.bindiff*), 282
binop_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1012
binop_to_function (*angr.engines.soot.expressions.binop.SimSootExpr_Binop attribute*), 1050
binop_to_function (*angr.engines.soot.expressions.SimSootExpr_Binop attribute*), 1047
BinOpNode (*class in angr.analyses.loop_analysis.loop_analysis*), 788
BinopOperatorEntry (*class in angr.protos.decompilation_cache_pb2*), 1402
binopv_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1012
bits (*angr.analyses.decompiler.known_patterns.dsl.PAny attribute*), 492
bits (*angr.analyses.decompiler.known_patterns.dsl.PConst*

*attribute*), 493
bits (*angr.analyses.decompiler.known_patterns.dsl.PExtract attribute*), 496
bits (*angr.analyses.decompiler.known_patterns.dsl.PPhi attribute*), 501
bits (*angr.analyses.decompiler.known_patterns.dsl.PVVar attribute*), 492
bits (*angr.analyses.decompiler.known_patterns.PAny attribute*), 464
bits (*angr.analyses.decompiler.known_patterns.PConst attribute*), 469
bits (*angr.analyses.decompiler.known_patterns.PExtract attribute*), 471
bits (*angr.analyses.decompiler.known_patterns.PPhi attribute*), 474
bits (*angr.analyses.decompiler.known_patterns.PVVar attribute*), 477
bits (*angr.analyses.propagator.values.Top property*), 798
bits (*angr.analyses.variable_recovery.engine_base.RichR property*), 924
bits (*angr.engines.light.data.RegisterOffset property*), 1010
bits (*angr.knowledge_plugins.key_definitions.atoms.Atom property*), 1242
bits (*angr.rust.sim_type.EnumVariant property*), 1434
bits (*angr.sim_variable.SimComboRegisterVariable property*), 1479
bits (*angr.sim_variable.SimVariable property*), 1478
BitwiseOrToLogicalOr (*class in angr.analyses.decompiler.peephole_optimizations.bitwise_or_to_logical_or*), 605
black (*angr.misc.ansi.BackgroundColor attribute*), 1314
black (*angr.misc.ansi.Color attribute*), 1314
Blade (*class in angr.blade*), 958
blank_state() (*angr.factory.AngrObjectFactory method*), 1110
Block (*angr.analyses.reaching_definitions.subject.SubjectType attribute*), 869
block (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1175
Block (*class in angr.block*), 960
Block (*class in angr.protos.primitives_pb2*), 1403
block() (*angr.analyses.decompiler.clinic.Clinic method*), 407
block() (*angr.factory.AngrObjectFactory method*), 1113
block() (*angr.sim_state.SimState method*), 1454
block_addr (*angr.ailment.Manager attribute*), 212
block_addr (*angr.ailment.manager.Manager attribute*), 224
block_addr (*angr.code_location.AILCodeLocation property*), 986
block_addrs (*angr.knowledge_plugins.functions.function.Function property*), 1212
block_addrs_set (*angr.knowledge_plugins.functions.function.Function property*), 1212
block_beginning_state() (*angr.knowledge_plugins.propagations.propagation_model.Propa method*), 1272
block_branch_ins_addr() (*in module angr.utils.vex*), 1703
block_count (*angr.state_plugins.history.SimStateHistory property*), 1562
block_id (*angr.analyses.cfg.cfg_emulated.CFGJob property*), 310
block_id (*angr.analyses.vfg.VFGJob property*), 932
block_id (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
block_idx (*angr.analyses.variable_recovery.engine_base.SimEngineVRBa property*), 924
block_is_single_instruction() (*in module angr.utils.vex*), 1703
block_key_to_addr() (*in module angr.knowledge_plugins.cfg.spilling_cfg*), 1181
block_key_to_size() (*in module angr.knowledge_plugins.cfg.spilling_cfg*), 1181
block_map (*angr.analyses.decompiler.known_patterns.finder.KnownPatter attribute*), 507
block_map (*angr.analyses.decompiler.known_patterns.KnownPatternMatch attribute*), 461
block_map (*angr.analyses.decompiler.KnownPatternMatch attribute*), 383
block_matches (*angr.analyses.bindiff.FunctionDiff property*), 281
BLOCK_MAX_SIZE (*angr.Block attribute*), 160
BLOCK_MAX_SIZE (*angr.block.Block attribute*), 960
block_similarity() (*angr.analyses.bindiff.FunctionDiff method*), 281
BlockCache (*class in angr.analyses.decompiler.clinic*), 403
BlockCache (*class in angr.analyses.decompiler.optimization_passes.optimi* 586
BlockDetails (*class in angr.state_plugins.unicorn_engine*), 1600
BlockGraph (*class in angr.protos.primitives_pb2*), 1403
BlockID (*class in angr.knowledge_plugins.cfg.block_id*), 1165
BlockIDProto (*class in angr.protos.cfg_pb2*), 1395
BlockIOFinder (*class in angr.analyses.decompiler.block_io_finder*), 398
BlockLocator (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter* 582
BlockNode (*class in angr.codenode*), 987
BlockPool (*class in angr.utils.ail_serialization*), 1671

BlockProtocol (*class in angr.engines.light.engine*), 1010
blocks (*angr.knowledge_plugins.functions.function.Function property*), 1211
blocks_by_addr (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass property*), 588
blocks_by_addr_and_idx (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass property*), 588
blocks_probably_identical() (*angr.analyses.bindiff.FunctionDiff method*), 281
BLOCKS_THRESHOLD (*angr.analyses.binary_optimizer.BinaryOptimizer attribute*), 279
BLOCKS_THRESHOLD (*angr.analyses.BinaryOptimizer attribute*), 237
blocks_with_differing_constants (*angr.analyses.bindiff.BinDiff property*), 282
blocks_with_differing_constants (*angr.analyses.bindiff.FunctionDiff property*), 281
BlockSimplifier (*class in angr.analyses.decompiler.block_simplifier*), 400
BlockStart (*class in angr.analyses.disassembly*), 740
BlockTerminationNotice, 1047
BlockWalker (*class in angr.analyses.decompiler.optimization_passes.condition_duplicator*), 562
BlockWalker (*class in angr.analyses.decompiler.optimization_passes.const_prop_reverter*), 563
blue (*angr.misc.ansi.BackgroundColor attribute*), 1314
blue (*angr.misc.ansi.Color attribute*), 1314
Bookmark (*class in angr.knowledge_plugins.bookmarks*), 1149
Bookmarks (*class in angr.knowledge_plugins.bookmarks*), 1149
BookmarksSerializer (*class in angr.angrdb.serializers.bookmarks*), 949
boolean_extend() (*in module angr.engines.vex.claripy.ccall*), 1068
boolean_operators_in_condition() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.duplication_reverter.DuplicationReverter static method*), 570
BooleanCounter (*class in angr.analyses.decompiler.counters.boolean_counter*), 411
booleanize() (*angr.engines.pcode.behavior.OpBehavior class method*), 1015
BooleanNaming (*class in angr.analyses.decompiler.semantic_naming.boolean_naming*), 647
BoolExprXor1 (*class in angr.analyses.decompiler.peephole_optimizations.bool_expr_xor_1*), 606
Boring (*angr.protos.primitives_pb2.Edge attribute*), 1402
BOTH (*angr.state_plugins.inspect.When attribute*), 1570
Bottom (*class in angr.analyses.propagator.values*), 798
BottomType (*class in angr.analyses.stack_pointer_tracker*), 893
BottomType (*class in angr.analyses.typehoon.typeconsts*), 907
BoyScout (*class in angr.analyses.boyscout*), 283
BP (*class in angr.state_plugins.inspect*), 1570
BP_AFTER (*angr.state_plugins.inspect.SimInspector attribute*), 1571
BP_AFTER (*angr.state_plugins.SimInspector attribute*), 1511
BP_BEFORE (*angr.state_plugins.inspect.SimInspector attribute*), 1571
BP_BEFORE (*angr.state_plugins.SimInspector attribute*), 1511
BP_BOTH (*angr.state_plugins.inspect.SimInspector attribute*), 1571
BP_BOTH (*angr.state_plugins.SimInspector attribute*), 1511
BP_IPDB() (*in module angr.state_plugins.inspect*), 1570
BP_IPYTHON() (*in module angr.state_plugins.inspect*), 1570
branches() (*angr.analyses.ddg.LiveDefinitions method*), 362
BreakNode (*class in angr.analyses.decompiler.structurer_nodes*), 712
BREAKPOINT (*angr.emulator.EmulatorStopReason attribute*), 996
BREAKPOINT (*angr.EmulatorStopReason attribute*), 161
breakpoints (*angr.emulator.Emulator property*), 995
brk (*class in angr.procedures.linux_kernel.brk*), 1364
bss_memory_read_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method*), 337
bss_memory_write_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.BSSHook method*), 337
BSSHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 337
Bswap (*class in angr.analyses.decompiler.peephole_optimizations.bswap*), 606
BTRflagsMergeBitExtraction (*class in angr.analyses.decompiler.peephole_optimizations.bt_rflags_merg*), 606
Bucketizer (*class in angr.exploration_techniques.bucketizer*), 1098
bug_report() (*in module angr.misc.bug_report*), 1315
build() (*angr.analyses.disassembly.Operand static*

*method*), 742
build() (*angr.rust.analyses.struct_builder.StructBuilder method*), 1410
build() (*angr.rust.optimization_passes.struct_instantiation_simplifier.StructBuilder method*), 1429
build_stmt_opts_by_kind() (*in module angr.analyses.decompiler.utils*), 724
bulk_import_serialized() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method*), 1180
bulk_import_serialized() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict method*), 1220
bulk_import_serialized() (*angr.knowledge_plugins.structured_code.SpillingDecompilationDict method*), 1279
bulk_import_serialized_nodes() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1183
bv_slice() (*in module angr.storage.memory_object*), 1667
bv_to_fp() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1587
BVS() (*angr.state_plugins.solver.SimSolver method*), 1589
byte_string (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
bytes (*angr.analyses.cfg.cfb.Unknown property*), 304
bytes (*angr.block.Block property*), 961
bytes_at() (*angr.storage.memory_object.SimMemoryObject method*), 1667
bzero (*class in angr.procedures.posix.bzero*), 1370

## C

c_args_as_atoms() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 844
c_repr() (*angr.analyses.decompiler.structured_codegen.c.CConstruct method*), 672
c_repr() (*angr.analyses.decompiler.structured_codegen.rust.RustConstruct method*), 692
c_repr() (*angr.analyses.typehoon.translator.SimTypeTempRef method*), 905
c_repr() (*angr.rust.sim_type.RustSimType method*), 1431
c_repr() (*angr.sim_type.SimStruct method*), 1471
c_repr() (*angr.sim_type.SimType method*), 1460
c_repr() (*angr.sim_type.SimTypeArray method*), 1467
c_repr() (*angr.sim_type.SimTypeBitfield method*), 1474
c_repr() (*angr.sim_type.SimTypeBottom method*), 1462
c_repr() (*angr.sim_type.SimTypeEnum method*), 1473
c_repr() (*angr.sim_type.SimTypeFixedSizeInt method*), 1464
c_repr() (*angr.sim_type.SimTypeFunction method*), 1469
c_repr() (*angr.sim_type.SimTypeInt method*), 1463
c_repr() (*angr.sim_type.SimTypePointer method*), 1466
c_repr() (*angr.sim_type.SimTypeRef method*), 1476
c_repr() (*angr.sim_type.SimTypeReference method*), 1467
c_repr() (*angr.sim_type.SimTypeString method*), 1468
c_repr() (*angr.sim_type.SimTypeWString method*), 1469
c_repr() (*angr.sim_type.SimUnion method*), 1472
c_repr() (*angr.sim_type.TypeRef method*), 1461
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CAILBlock method*), 674
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CAssig method*), 676
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CBina method*), 681
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CBrea method*), 675
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCons method*), 682
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCons method*), 672
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CCont method*), 676
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDirty method*), 684
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDirty method*), 678
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CDoW method*), 674
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CExpr method*), 677
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFake method*), 679
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CForL method*), 675
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunc method*), 673
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunc method*), 677
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CGoto method*), 678
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfBre method*), 675
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIfEls method*), 675
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIncon method*), 676
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CIndex method*), 680
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CITE method*), 682
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CLabe method*), 678

c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 683
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 682
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 677
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 674
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 679
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 676
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 681
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 680
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 678
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 679
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 680
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 683
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 683
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 674
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 694
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 700
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 697
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 703
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 696
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 704
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 692
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 696
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 706
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 699
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 694
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 700
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 701
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 695
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 693
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 698
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 697
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 698
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 695
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 695
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 697
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 699
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 702
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 694
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 705
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 699
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 701
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 705
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 696
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 704
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 698
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 693
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 701
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 700
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 700
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 696
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 703
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 702
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 698
c_repr_chunks() (angr.analyses.decompiler.structured_codegen.rust.RustCodeGenerator method), 701

c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.rust.RustVariableFields method*), 702
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.rust.RustVectorConvert method*), 706
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.rust.RustVEXCCallExpression method*), 705
c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.rust.RustWhileLoop method*), 694
c_return_as_atoms() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 845
cache_limit (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1186
cache_limit (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict property*), 1180
cache_limit (*angr.knowledge_plugins.functions.function_manager.FunctionManager property*), 1227
cache_limit (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict property*), 1220
cache_limit (*angr.knowledge_plugins.structured_code.SpillingDecompilationDict property*), 1279
cache_limit (*angr.knowledge_plugins.variables.spilling_vardict.SpillingVariableInternalDict property*), 1282
cache_state() (*angr.analyses.pathfinder.Pathfinder method*), 795
cached_count (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1186
cached_count (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict property*), 1180
cached_count (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict property*), 1220
cached_function_count (*angr.knowledge_plugins.functions.function_manager.FunctionManager property*), 1227
cached_keys (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict property*), 1220
CAILBlock (*class in angr.analyses.decompiler.structured_codegen.c*), 674
CAILBlockMsg (*class in angr.protos.codegen_pb2*), 1397
calc_paritybit() (*in module angr.engines.vex.claripy.ccall*), 1068
calc_size() (*angr.calling_conventions.AllocHelper class method*), 963
calc_zerobit() (*in module angr.engines.vex.claripy.ccall*), 1068
calculate() (*angr.engines.vex.claripy.irop.SimIROp method*), 1074
calculate_iterated_dominace_frontier_set() (*in module angr.analyses.dominance_frontier*), 744
calculate_stack_depth() (*angr.analyses.decompiler.clinic.Clinic method*), 407
Call (*angr.protos.primitives_pb2.Edge attribute*), 1403
CALL (*angr.rustvariablefields.inspect.EventType attribute*), 1565
Call (*class in angr.vectorConvert expression*), 218
call() (*angr.sim_procedure.SimProcedure method*), 1451
call() (*angr.state_plugins.callstack.CallStack method*), 1543
call_c() (*angr.callable.Callable method*), 962
call_clobbered_regs() (*in module angr.calling_conventions*), 984
call_destination_keys (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1185
call_destination_nodes() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1185
call_exprs_in_graph() (*in module angr.analyses.decompiler.utils*), 726
call_stack (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 1326
call_stack_copy() (*angr.analyses.cfg.cfg_job_base.CFGJobBase method*), 1326
call_state() (*angr.factory.AngrObjectFactory method*), 1111
call_target_fn (*angr.analyses.decompiler.known_patterns.dsl.MatchCtx attribute*), 491
call_target_names() (*angr.analyses.decompiler.known_patterns.finder.KnownPatternF method*), 509
callable (*angr.knowledge_plugins.functions.function.Function property*), 1217
Callable (*class in angr.callable*), 961
callable() (*angr.factory.AngrObjectFactory method*), 1112
CallAnalysis (*class in angr.analyses.vfg*), 933
CallBooleanMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallBooleanMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
CallByteMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallByteMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
CallCharMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallCharMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
CALLEE_CLEANUP (*angr.calling_conventions.SimCC at-*

*tribute*), 967
CALLEE_CLEANUP (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 972
CALLEE_CLEANUP (*angr.calling_conventions.SimCCStdcall attribute*), 972
CALLEE_CLEANUP (*angr.SimCC attribute*), 173
callee_return (*angr.analyses.purity.engine.DataSource attribute*), 801
callee_return_name (*angr.analyses.purity.engine.DataSource property*), 801
CalleeCleanupFinder (*class in angr.analyses.callee_cleanup_finder*), 283
CalleeWriteCollector (*class in angr.rust.analyses.rust_calling_convention.fact_collector*), 1408
Caller (*class in angr.procedures.stubs.caller*), 1378
CALLER_SAVED_REGS (*angr.calling_conventions.SimCC attribute*), 967
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 976
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCARM attribute*), 977
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 977
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCCdecl attribute*), 971
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCGoAMD64ABIInternal attribute*), 976
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 972
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 972
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCN64 attribute*), 980
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCO32 attribute*), 979
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCRISCV64 attribute*), 979
CALLER_SAVED_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
CALLER_SAVED_REGS (*angr.SimCC attribute*), 173
caller_saved_regs_as_atoms() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler static method*), 845
caller_will_handle_single_ret (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 840
caller_will_handle_single_ret (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
CallExprFinder (*class in angr.knowledge_plugins.propagations.states*), 1272
CallFinder (*class in angr.rust.utils.ail*), 1437
CallFunctionGoal (*class in angr.exploration_techniques.director*), 1100
callgraph (*angr.knowledge_base.KnowledgeBase property*), 1119
CallGraphSerializer (*class in angr.angrdb.serializers.callgraph*), 950
CallGraphVisitor (*class in angr.analyses.forward_analysis.visitors.call_graph*), 766
calling_convention (*angr.knowledge_plugins.functions.function.Function property*), 1211
calling_convention() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
CallingConventionAnalysis (*class in angr.analyses.calling_convention.calling_convention*), 285
CallingConventionAnalysisMode (*class in angr.analyses.complete_calling_conventions*), 353
CallingConventionSerializer (*class in angr.knowledge_plugins.functions.function_parser*), 1227
CallIntMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallIntMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
CallLongMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallLongMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
CallMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallMethodA (*class in angr.procedures.java_jni.method_calls*), 1342
CallMethodBase (*class in angr.procedures.java_jni.method_calls*), 1342
CallNonvirtualBooleanMethod (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualBooleanMethodA (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualByteMethod (*class in*

*angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualByteMethodA (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualCharMethod (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualCharMethodA (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualIntMethod (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualIntMethodA (*class in angr.procedures.java_jni.method_calls*), 1345
CallNonvirtualLongMethod (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualLongMethodA (*class in angr.procedures.java_jni.method_calls*), 1345
CallNonvirtualMethod (*class in angr.procedures.java_jni.method_calls*), 1343
CallNonvirtualMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
CallNonvirtualObjectMethod (*class in angr.procedures.java_jni.method_calls*), 1343
CallNonvirtualObjectMethodA (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualShortMethod (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualShortMethodA (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualVoidMethod (*class in angr.procedures.java_jni.method_calls*), 1344
CallNonvirtualVoidMethodA (*class in angr.procedures.java_jni.method_calls*), 1345
CallObjectMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallObjectMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
calloc (*class in angr.analyses.identifier.functions.skip_calloc*), 778
calloc (*class in angr.procedures.libc.calloc*), 1353
calloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 1558
callout_sites (*angr.knowledge_plugins.functions.function.Function property*), 1213
CallProxiNode (*class in angr.analyses.proximity_graph*), 799
CallReached, 1003
CallResultNaming (*class in angr.analyses.decompiler.semantic_naming.call_result_naming*), 648
CallReturn (*class in angr.procedures.stubs.CallReturn*), 1376
CallRewriter (*class in angr.rust.optimization_passes.utils*), 1431
calls_in_graph() (*in module angr.analyses.decompiler.utils*), 725
CallShortMethod (*class in angr.procedures.java_jni.method_calls*), 1342
CallShortMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
CallSite (*class in angr.analyses.reaching_definitions.call_trace*), 834
CallSite (*class in angr.protos.function_pb2*), 1402
CALLSITE_DECOMPILER (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 1209
CALLSITE_DECOMPILER (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
CALLSITE_LOW (*angr.knowledge_plugins.functions.function.PrototypeSourc attribute*), 1209
CALLSITE_LOW (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
callsite_repr() (*angr.analyses.cfg.cfg_job_base.FunctionKey method*), 326
callsite_repr() (*angr.knowledge_plugins.cfg.block_id.BlockID method*), 1165
CallSiteFact (*class in angr.analyses.calling_convention.calling_convention*), 285
CallsiteFactCollector (*class in angr.rust.analyses.rust_calling_convention.fact_collector*), 1408
CallSiteMaker (*class in angr.analyses.decompiler.callsite_maker*), 401
CallsitePrototypeKind (*class in angr.knowledge_plugins.callsite_prototypes*), 1150
CallsitePrototypes (*class in*

*angr.knowledge_plugins.callsite_prototypes*), 1150
callsites_to() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 869
CallStack (*class in angr.state_plugins.callstack*), 1542
callstack_key (*angr.knowledge_plugins.cfg.cfg_node.CFGENode property*), 1175
callstack_key (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
callstack_repr() (*angr.analyses.vfg.VFGJob method*), 932
CallStackAction (*class in angr.state_plugins.callstack*), 1544
CallStatementRewriter (*class in angr.analyses.decompiler.optimization_passes.call_stmt_rewriter*), 560
CallStaticBooleanMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticBooleanMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticByteMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticByteMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticCharMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticCharMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticIntMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticIntMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticLongMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticLongMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticMethodA (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticObjectMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticObjectMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticShortMethod (*class in angr.procedures.java_jni.method_calls*), 1345
CallStaticShortMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticVoidMethod (*class in angr.procedures.java_jni.method_calls*), 1346
CallStaticVoidMethodA (*class in angr.procedures.java_jni.method_calls*), 1346
CallTrace (*angr.analyses.reaching_definitions.subject.SubjectType attribute*), 869
CallTrace (*class in angr.analyses.reaching_definitions.call_trace*), 834
CallTracingFilter (*class in angr.analyses.veritesting*), 930
CallVisitor (*class in angr.rust.utils.ail*), 1438
CallVoidMethod (*class in angr.procedures.java_jni.method_calls*), 1343
CallVoidMethodA (*class in angr.procedures.java_jni.method_calls*), 1343
can_call_other_funcs() (*angr.analyses.identifier.func.Func method*), 771
can_call_other_funcs() (*angr.analyses.identifier.functions.memcmp.memcmp method*), 775
can_call_other_funcs() (*angr.analyses.identifier.functions.memcpy.memcpy method*), 775
can_call_other_funcs() (*angr.analyses.identifier.functions.memset.memset method*), 776
can_call_other_funcs() (*angr.analyses.identifier.functions.strcasecmp.strcasecmp method*), 781
can_call_other_funcs() (*angr.analyses.identifier.functions.strcmp.strcmp method*), 782
can_call_other_funcs() (*angr.analyses.identifier.functions.strcpy.strcpy method*), 782
can_call_other_funcs() (*angr.analyses.identifier.functions.strncmp.strncmp method*), 783
can_call_other_funcs()

(*angr.analyses.identifier.functions.strncpy.strncpy method*), 784
can_call_same_name() (*angr.analyses.identifier.identify.Identifier method*), 785
can_swap() (*angr.analyses.decompiler.block_io_finder.BlockIOFinder method*), 398
canonical_form() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 886
capstone (*angr.block.Block property*), 961
CapstoneBlock (*class in angr.block*), 959
CapstoneInsn (*class in angr.block*), 960
cardinality (*angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor.AbstractAddressDescriptor property*), 1659
CArrayTypeLength (*class in angr.analyses.decompiler.structured_codegen.c*), 684
CAS (*class in angr.ailment.statement*), 224
cas_intrinsic_name() (*in module angr.analyses.decompiler.peephole_optimizations.cas_intrinsics*), 607
CascadingConditionNode (*class in angr.analyses.decompiler.structurer_nodes*), 712
CascadingConditionTransformer (*class in angr.analyses.decompiler.region_simplifiers.cascading_ifs*), 633
CascadingIfsRemover (*class in angr.analyses.decompiler.region_simplifiers.cascading_ifs*), 633
Case (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier*), 584
Case2Result (*class in angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 338
cases_issubset() (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier static method*), 585
CASIntrinsics (*class in angr.analyses.decompiler.peephole_optimizations.cas_intrinsics*), 607
CAssignment (*class in angr.analyses.decompiler.structured_codegen.c*), 676
CAssignmentMsg (*class in angr.protos.codegen_pb2*), 1397
cast_primitive() (*angr.simos.javavm.SimJavaVM static method*), 1487
cast_to_mem() (*angr.analyses.data_dep.dep_nodes.MemDepNode class method*), 360
CastType (*angr.state_plugins.solver.SimSolver attribute*), 1591
CatchDesync (*angr.exploration_techniques.tracer.TracingMode attribute*), 1107
categories (*angr.analyses.decompiler.known_patterns.dslPVVar attribute*), 492
categories (*angr.analyses.decompiler.known_patterns.PVVar attribute*), 477
category (*angr.storage.file.SimFile property*), 1612
category (*angr.storage.memory_mixins.memory_mixin.MemoryMixin property*), 1638
category (*angr.storage.memory_mixins.regioned_memory.region_categor property*), 1659
CBinaryOp (*class in angr.analyses.decompiler.structured_codegen.c*), 680
CBinaryOpMsg (*class in angr.protos.codegen_pb2*), 1397
CBreak (*class in angr.analyses.decompiler.structured_codegen.c*), 675
CBreakMsg (*class in angr.protos.codegen_pb2*), 1397
cbSize (*angr.procedures.win32.GetLastInputInfo.GetLastInputInfo attribute*), 1382
cc (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
cc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
cc (*angr.analyses.reaching_definitions.subject.Subject property*), 870
cc() (*angr.factory.AngrObjectFactory method*), 1113
CCA_DECOMPILER (*angr.knowledge_plugins.functions.function.PrototypeSo attribute*), 1209
CCA_DECOMPILER (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
CCA_LOW (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 1209
CCA_LOW (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
ccall_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1012
CCallMultivaluedException, 1068
CCallRewriterBase (*class in angr.analyses.decompiler.ccall_rewriters.base*), 402
CCallTables (*class in angr.engines.vex.claripy.ccall*), 1068
CClosingObject (*class in angr.analyses.decompiler.structured_codegen.c*), 684
CCondPropBlockWalker (*class in angr.analyses.decompiler.optimization_passes.condition_constpro* 562
CConstant (*class in angr.analyses.decompiler.structured_codegen.c*), 681
CConstantMsg (*class in angr.protos.codegen_pb2*), 1397
CConstantReferenceValue (*class in angr.protos.codegen_pb2*), 1397
CConstantType (*class in angr.analyses.decompiler.structured_codegen.base*), 668
CConstruct (*class in angr.analyses.decompiler.structured_codegen.c*),

672
CConstructNode (*class in angr.protos.codegen_pb2*), 1397
CConstructTags (*class in angr.protos.codegen_pb2*), 1398
CConstructTags.JsonValuesEntry (*class in angr.protos.codegen_pb2*), 1398
CContinue (*class in angr.analyses.decompiler.structured_codegen.c*), 675
CContinueMsg (*class in angr.protos.codegen_pb2*), 1398
CDG (*class in angr.analyses.cdg*), 289
CDirtyExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 683
CDirtyExpressionMsg (*class in angr.protos.codegen_pb2*), 1398
CDirtyStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 678
CDirtyStatementMsg (*class in angr.protos.codegen_pb2*), 1398
CDoWhileLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 674
ceil_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1169
ceiling_addr() (*angr.analyses.cfg.cfb.CFBlanket method*), 305
ceiling_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionDict method*), 1219
ceiling_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1218
ceiling_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManagerBase method*), 1222
ceiling_func() (*angr.knowledge_plugins.functions.function_manager.FunctionManagerBase method*), 1223
ceiling_item() (*angr.analyses.cfb.CFBlanket method*), 305
ceiling_items() (*angr.analyses.cfb.CFBlanket method*), 306
CExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 673
CExpressionStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 676
CExpressionStatementMsg (*class in angr.protos.codegen_pb2*), 1398
cextern_sort_key() (*in module angr.analyses.decompiler.structured_codegen.c*), 671
CFakeVariable (*class in angr.analyses.decompiler.structured_codegen.c*), 679
CFakeVariableMsg (*class in angr.protos.codegen_pb2*), 1398
CFAMixin (*class in angr.rust.mixins.cfa_mixin*), 1417
CFBlanket (*class in angr.analyses.cfg.cfb*), 305
CFBlanketView (*class in angr.analyses.cfg.cfb*), 304
cfg (*angr.mcp.ProjectSession attribute*), 1297
cfg (*angr.mcp.session.ProjectSession attribute*), 1307
CFG (*class in angr.analyses.cfg.cfg*), 306
CFG (*class in angr.protos.cfg_pb2*), 1395
cfg_cache (*angr.analyses.Veritesting attribute*), 272
cfg_cache (*angr.analyses.veritesting.CallTracingFilter attribute*), 931
cfg_cache (*angr.analyses.veritesting.Veritesting attribute*), 931
CFG_HANDLE_JOB (*angr.state_plugins.inspect.EventType attribute*), 1565
cfg_jumpkind_from_pb() (*in module angr.utils.enums_conv*), 1679
cfg_jumpkind_to_pb() (*in module angr.utils.enums_conv*), 1679
CFGArchOptions (*class in angr.analyses.cfg.cfg_arch_options*), 307
CFGBase (*class in angr.analyses.cfg.cfg_base*), 307
CFGEdgeData (*class in angr.protos.cfg_pb2*), 1395
CFGEmulated (*class in angr.analyses.cfg.cfg_emulated*), 311
CFGENode (*class in angr.knowledge_plugins.cfg.cfg_node*), 1175
CFGENodeData (*class in angr.protos.cfg_pb2*), 1395
CFGFast (*class in angr.analyses.cfg.cfg_fast*), 320
CFGFastSoot (*class in angr.analyses.cfg.cfg_fast_soot*), 325
CFGJob (*class in angr.analyses.cfg.cfg_emulated*), 310
CFGJob (*class in angr.analyses.cfg.cfg_fast*), 320
CFGJobBase (*class in angr.analyses.cfg.cfg_job_base*), 326
CFGJobType (*class in angr.analyses.cfg.cfg_fast*), 319
CFGManager (*class in angr.knowledge_plugins.cfg.cfg_manager*), 1165
CFGModel (*class in angr.knowledge_plugins.cfg.cfg_model*), 1166
CFGModelSerializer (*class in angr.angrdb.serializers.cfg_model*), 951
CFGNode (*class in angr.knowledge_plugins.cfg.cfg_node*), 1173
CFGNode (*class in angr.protos.cfg_pb2*), 1395
CFGNodeCreationFailure (*class in angr.knowledge_plugins.cfg.cfg_node*), 1173
CFGNotBuiltError, 1301
CFGResumeState (*class in angr.analyses.cfg.cfg_fast*), 318
CFGSliceToSink (*class in angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink*), 348

CFGTransformationMixin (*class in angr.rust.mixins.cfg_transformation_mixin*), 1418
CForLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 674
CForLoopMsg (*class in angr.protos.codegen_pb2*), 1398
cfuncall_config() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 689
CFunction (*class in angr.analyses.decompiler.structured_codegen.c*), 672
CFunctionCall (*class in angr.analyses.decompiler.structured_codegen.c*), 677
CFunctionCallConfigEntry (*class in angr.protos.codegen_pb2*), 1398
CFunctionCallMsg (*class in angr.protos.codegen_pb2*), 1398
CFunctionMsg (*class in angr.protos.codegen_pb2*), 1398
CGoto (*class in angr.analyses.decompiler.structured_codegen.c*), 677
CGotoMsg (*class in angr.protos.codegen_pb2*), 1398
ChainMapCOW (*class in angr.utils.cowdict*), 1676
ChallRespInfo (*class in angr.state_plugins.trace_additions*), 1597
changed_bytes() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1642
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage method*), 1648
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.multi_page.MultiPage method*), 1649
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 1651
changed_bytes() (*angr.storage.memory_mixins.paged_memory.pages.treap_page.TreapPage method*), 1655
changed_bytes() (*angr.storage.memory_mixins.slotted_memory_mixin.SlottedMemoryMixin method*), 1665
changed_pages() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1642
CharacterIsDigit (*class in angr.procedures.java_lang.character*), 1348
CharacterIsSpaceChar (*class in angr.procedures.java_lang.character*), 1348
CharNextA (*class in angr.procedures.win_user32.chars*), 1387
CharPrevA (*class in angr.procedures.win_user32.chars*), 1387
chase_fn (*angr.analyses.decompiler.known_patterns.dsl.MatchCtx attribute*), 491
chased_defs (*angr.analyses.decompiler.known_patterns.dsl.MatchCtx attribute*), 490
chdir (*class in angr.procedures.linux_kernel.cwd*), 1365
chdir() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
check() (*angr.exploration_techniques.director.BaseGoal method*), 1099
check() (*angr.exploration_techniques.director.CallFunctionGoal method*), 1100
check() (*angr.exploration_techniques.director.ExecuteAddressGoal method*), 1099
check() (*angr.state_plugins.inspect.BP method*), 1570
check_array_bounds() (*angr.engines.soot.values.arrayref.SimSootValue_ArrayRef static method*), 1059
check_in_between_stmts() (*in module angr.utils.ssa*), 1695
check_offset() (*angr.calling_conventions.SimRegArg method*), 964
check_state() (*angr.exploration_techniques.director.BaseGoal method*), 1099
check_state() (*angr.exploration_techniques.director.CallFunctionGoal method*), 1100
check_state() (*angr.exploration_techniques.director.ExecuteAddressGoal method*), 1100
check_tests() (*angr.analyses.identifier.identify.Identifier method*), 785
check_value_get() (*angr.calling_conventions.SimFunctionArgument method*), 964
check_value_set() (*angr.calling_conventions.SimFunctionArgument method*), 964
checkpoint() (*angr.analyses.decompiler.region_overlay.OverlayManager method*), 624
chop_value() (*angr.knowledge_plugins.propagations.prop_value.PropValue static method*), 1269
chroot (*class in angr.procedures.posix.chroot*), 1371
chroot() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
Chunk (*class in angr.state_plugins.heap.heap_freelist*), 1557
chunk_from_mem() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 1558
chunk_from_mem() (*angr.state_plugins.heap.heap_ptmalloc.SimHeapPTMalloc method*), 1560
chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 1558
CIfBreak (*class in angr.analyses.decompiler.structured_codegen.c*), 675
CIfBreakMsg (*class in angr.protos.codegen_pb2*), 1399
CIfElse (*class in angr.analyses.decompiler.structured_codegen.c*), 675
CIfElseBranch (*class in angr.protos.codegen_pb2*), 1399
CIfElseMsg (*class in angr.protos.codegen_pb2*), 1399
CIncompleteSwitchCase (*class in angr.analyses.decompiler.structured_codegen.c*), 676
CIncompleteSwitchCaseEntry (*class in angr.protos.codegen_pb2*), 1399
CIncompleteSwitchCaseMsg (*class in*

*angr.protos.codegen_pb2*), 1399
CIndexedVariable (*class in angr.analyses.decompiler.structured_codegen.c*), 679
CIndexedVariableMsg (*class in angr.protos.codegen_pb2*), 1399
CITE (*class in angr.analyses.decompiler.structured_codegen.c*), 682
CITEMsg (*class in angr.protos.codegen_pb2*), 1399
CLabel (*class in angr.analyses.decompiler.structured_codegen.c*), 678
CLabelMsg (*class in angr.protos.codegen_pb2*), 1399
claims_only (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 458
claims_only (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521
claims_only (*angr.analyses.decompiler.KnownPattern attribute*), 379
claripy_ast_from_ail_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 409
claripy_ast_to_sympy_expr() (*angr.analyses.decompiler.condition_processor.ConditionProcessor static method*), 410
ClaripyDataEngineMixin (*class in angr.analyses.propagator.top_checker_mixin*), 797
ClaripyDataMixin (*class in angr.engines.vex.claripy.datalayer*), 1073
ClaripyDataVEXEngineMixin (*class in angr.analyses.propagator.top_checker_mixin*), 797
ClassIdentifier (*class in angr.analyses.class_identifier*), 351
clean() (*angr.utils.cowdict.ChainMapCOW method*), 1676
clean() (*angr.utils.cowdict.DefaultChainMapCOW method*), 1676
cleanup() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 316
cleanup() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 685
cleanup() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeGenerator method*), 707
cleanup() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method*), 1278
cleanup_callee_graph() (*angr.analyses.outliner.outliner.Outliner method*), 792
CleanupCodeRemover (*class in angr.rust.optimization_passes.cleanup_code_remover*), 1421
CleanupFunctionIdentification (*class in angr.rust.analyses.cleanup_function_identification*), 1406
clear() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 408
clear() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method*), 1180
clear() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1221
clear() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionManager method*), 1220
clear() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
clear() (*angr.state_plugins.log.SimStateLog method*), 1577
clear() (*angr.state_plugins.scratch.SimStateScratch method*), 1584
clear() (*angr.storage.memory_mixins.regioned_memory.abstract_address.AbstractAddress method*), 1659
clear_cache() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 1044
clear_cache() (*angr.engines.vex.lifter.VEXLifter method*), 1079
clear_hash() (*angr.ailment.Block method*), 210
clear_hash() (*angr.ailment.block.Block method*), 214
clear_hash() (*angr.sim_variable.SimVariable method*), 1478
clear_local_references() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 1574
clear_page_cache() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
clear_range() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 1653
clear_region_for_reflow() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1173
clear_updated_functions() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 316
ClemoryBackerMixin (*class in angr.storage.memory_mixins.paged_memory.page_backer_mixins*), 1641
ClientReq (*angr.protos.primitives_pb2.Edge attribute*), 1408
Clinic (*class in angr.analyses.decompiler.clinic*), 405
Clinic (*class in angr.protos.clinic_pb2*), 1396
Clinic.InlinedCountsEntry (*class in angr.protos.clinic_pb2*), 1396
Clinic.StackItemsEntry (*class in angr.protos.clinic_pb2*), 1396
Clinic.VvarToVvarEntry (*class in angr.protos.clinic_pb2*), 1396
ClinicFactory (*class in angr.rust.knowledge_plugins.clinic_factory*), 1412

ClinicMode (*class in angr.analyses.decompiler.clinic*), 403
ClinicNamingBase (*class in angr.analyses.decompiler.semantic_naming.naming_base*), 649
ClinicStage (*class in angr.analyses.decompiler.clinic*), 404
clock_gettime (*class in angr.procedures.posix.sim_time*), 1375
clone_graph_replace_splits() (*angr.analyses.decompiler.optimization_passes.duplicate_expr_comments.ExprCommentsEntry static method*), 569
CLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 674
CLoopMsg (*class in angr.protos.codegen_pb2*), 1399
close (*class in angr.procedures.posix.close*), 1371
close() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
close() (*angr.vaults.Vault static method*), 1704
close() (*angr.vaults.VaultShelf method*), 1705
close_project() (*in module angr.mcp.server*), 1307
close_session() (*angr.mcp.session.SessionManager method*), 1309
closed_fds (*angr.state_plugins.posix.SimSystemPosix property*), 1581
closedir (*class in angr.procedures.posix.closedir*), 1371
closelog (*class in angr.procedures.libc.closelog*), 1353
closest_common_ancestor() (*angr.state_plugins.history.SimStateHistory method*), 1562
Closure (*class in angr.utils.mp*), 1693
cls_ (*in module angr.calling_conventions*), 984
CmpMaskedShift (*class in angr.analyses.decompiler.peephole_optimizations.cmp_tagging_shifts*), 608
CmpOp (*class in angr.analyses.decompiler.region_simplifiers.switch_cases.simplifier*), 639
CmpORDRewriter (*class in angr.analyses.decompiler.peephole_optimizations.cmpord_rewriter*), 609
CmpSubConst (*class in angr.analyses.decompiler.peephole_optimizations.cmp_sub_const*), 608
CMultiStatementExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 682
CMultiStatementExpressionMsg (*class in angr.protos.codegen_pb2*), 1399
CoalesceAdjacentShiftRights (*class in angr.analyses.decompiler.peephole_optimizations.coalesce_adjacent_shrs*), 609
CoalesceSameCascadingIfs (*class in angr.analyses.decompiler.peephole_optimizations.coalesce_same_cascading_ifs*), 609
code_constants (*angr.knowledge_plugins.functions.function.Function property*), 1213
code_nodes (*angr.knowledge_plugins.functions.function.Function property*), 1211
CodeCave (*class in angr.analyses.codecave*), 352
CodeCaveAnalysis (*class in angr.analyses.codecave*), 352
CodeCaveClassification (*class in angr.analyses.codecave*), 352
Codegen (*class in angr.protos.codegen_pb2*), 1400
Codegen.ExprCommentsEntry (*class in angr.protos.codegen_pb2*), 1401
Codegen.NotesJsonEntry (*class in angr.protos.codegen_pb2*), 1401
Codegen.StmtCommentsEntry (*class in angr.protos.codegen_pb2*), 1401
CodeLocation (*class in angr.code_location*), 985
CodeMotionOptimization (*class in angr.analyses.decompiler.optimization_passes.code_motion*), 560
codenode (*angr.block.Block property*), 961
codenode (*angr.block.SootBlock property*), 961
CodeNode (*class in angr.analyses.decompiler.structurer_nodes*), 711
CodeNode (*class in angr.codenode*), 987
CodeReference (*angr.knowledge_plugins.cfg.memory_data.MemoryDataS attribute*), 1177
CodeReference (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
CodeReference (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
CodeReference (*class in angr.protos.primitives_pb2*), 1403
CodeTagging (*class in angr.analyses.code_tagging*), 351
CodeTags (*class in angr.analyses.code_tagging*), 351
collapse() (*angr.analyses.decompiler.edits.hooks*), 442
collapse() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecas classmethod*), 687
collapse() (*angr.analyses.decompiler.structured_codegen.rust.MakeType class method*), 709
collapse_capture (*angr.analyses.decompiler.known_patterns.KnownPatt attribute*), 457
collapse_capture (*angr.analyses.decompiler.known_patterns.pattern.Kn attribute*), 521
collapse_capture (*angr.analyses.decompiler.KnownPattern attribute*), 379
collapse_max_capture (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 457
collapse_max_capture (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521
collapse_max_capture

(*angr.analyses.decompiler.KnownPattern attribute*), 379
collapse_to() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 631
collect() (*angr.rust.analyses.rust_calling_convention.fact_collector.CallerWriteCollector method*), 1408
collect() (*angr.rust.analyses.rust_calling_convention.fact_collector.CallsiteFactCollector method*), 1408
collect() (*angr.rust.analyses.rust_calling_convention.fact_collector.CompRetValueCollector method*), 1408
collect() (*angr.rust.analyses.rust_calling_convention.fact_collector.FactCollector method*), 1409
collect() (*angr.rust.analyses.rust_calling_convention.fact_collector.MemoryWriteCollector method*), 1408
collect_callsite_stack_defs() (*angr.rust.mixins.dfa_mixin.DFAMixin method*), 1419
collect_conditions_between_nodes() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.duplication_reverter.DuplicationReverter method*), 571
COLLECT_DATA_REFS (*angr.analyses.decompiler.clinic.ClinicMode attribute*), 403
COLLECT_DATA_REFS (*angr.analyses.decompiler.ClinicMode attribute*), 373
COLLECT_EXTERNS (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 405
collect_ret_expr() (*angr.rust.optimization_passes.struct_return_simplifier.StructReturnSimplifier method*), 1430
collect_stack_defs_at() (*angr.rust.mixins.dfa_mixin.DFAMixin method*), 1419
Color (*class in angr.misc.ansi*), 1314
color() (*angr.analyses.disassembly.DisassemblyPiece static method*), 740
color() (*in module angr.misc.ansi*), 1314
ComboRegister (*class in angr.ailment.expression*), 219
ComboRegisterRewriter (*class in angr.rust.optimization_passes.combo_register_rewriter*), 1421
ComboRegReferenceWalker (*class in angr.analyses.decompiler.clinic*), 405
Comment (*class in angr.analyses.disassembly*), 742
Comment (*class in angr.knowledge_plugins.comments*), 1189
CommentKind (*class in angr.knowledge_plugins.comments*), 1189
Comments (*class in angr.knowledge_plugins.comments*), 1190
CommentsSerializer (*class in angr.angrdb.serializers.comments*), 951
commit() (*angr.analyses.decompiler.region_overlay.OverlayManager method*), 624
commutative (*angr.analyses.decompiler.known_patterns.dscompiler attribute*), 494
commutative (*angr.analyses.decompiler.known_patterns.PBinOp attribute*), 465
compare() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinition method*), 863
compare() (*angr.knowledge_plugins.key_definitions.environment.Environm method*), 1248
compare() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveD method*), 1252
compare() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1639
compare() (*angr.storage.memory_mixins.paged_memory.paged_memory_ method*), 1642
compare() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_pa method*), 1651
compare_path_group() (*angr.analyses.congruency_check.CongruencyCheck method*), 355
compare_paths() (*angr.analyses.congruency_check.CongruencyCheck method*), 355
compare_statement_dict() (*in module angr.analyses.bindiff*), 280
compare_states() (*angr.analyses.congruency_check.CongruencyCheck method*), 355
comparison_info (*angr.utils.balancer.Balancer attribute*), 1673
COMPARISON_NEGATION (*angr.ailment.BinaryOp attribute*), 209
COMPARISON_NEGATION (*angr.ailment.expression.BinaryOp attribute*), 218
compat_ret (*angr.utils.balancer.Balancer property*), 1673
compiler (*angr.analyses.language_detector.LanguageDetectionResult attribute*), 786
compiler (*angr.analyses.language_detector.LanguageDetector property*), 787
compiler_version (*angr.analyses.language_detector.LanguageDetection attribute*), 786
compiler_version (*angr.analyses.language_detector.LanguageDetector property*), 787
complete() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1098
complete() (*angr.sim_manager.SimulationManager method*), 1443
complete_graph_entry_count() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 627
COMPLETE_SCANNING (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 319
CompleteCallingConventionsAnalysis (*class in angr.analyses.complete_calling_conventions*), 353
complete() (*angr.llm_client.LLMClient method*), 1294

completion_json() (*angr.llm_client.LLMClient method*), 1294
completion_structured() (*angr.llm_client.LLMClient method*), 1294
complexity_error (*angr.analyses.decompiler.decompiler.Decompiler property*), 415
compute() (*angr.state_plugins.trace_additions.FormatInfo method*), 1596
compute() (*angr.state_plugins.trace_additions.FormatInfoBinaryOp method*), 1597
compute() (*angr.state_plugins.trace_additions.FormatInfoConcrete method*), 1596
compute() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 1596
compute_common_type() (*angr.analyses.decompiler.structured_codegen.c.CConstant static method*), 680
compute_common_type() (*angr.analyses.decompiler.structured_codegen.rust.RustBinaryOp static method*), 703
compute_dominance_frontier() (*in module angr.utils.graph*), 1684
compute_dominance_intervals() (*in module angr.utils.graph*), 1684
compute_quotient_graph() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 903
concat() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 1650
ConcatSimplifier (*class in angr.analyses.decompiler.peephole_optimizations.concat_simplifier*), 610
concrete (*angr.state_plugins.debug_variables.SimDebugVariable property*), 1546
concrete (*angr.state_plugins.view.SimMemView property*), 1607
concrete_bytes() (*angr.storage.memory_object.SimMemoryObject method*), 1667
concrete_load() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1639
concrete_path_bool() (*in module angr.state_plugins.solver*), 1587
concrete_path_list() (*in module angr.state_plugins.solver*), 1587
concrete_path_not_bool() (*in module angr.state_plugins.solver*), 1587
concrete_path_scalar() (*in module angr.state_plugins.solver*), 1587
concrete_path_tuple() (*in module angr.state_plugins.solver*), 1587
concrete_run_length() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1640
concrete_run_length() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 1655
concrete_states (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery property*), 925
concrete_variables() (*in module angr.analyses.decompiler.edits.resolve*), 448
concretize() (*angr.analyses.typehoon.translator.TypeTranslator method*), 906
concretize() (*angr.concretization_strategies.base.SimConcretizationStrategy method*), 991
concretize() (*angr.storage.file.SimFile method*), 1612
concretize() (*angr.storage.file.SimFileBase method*), 1610
concretize() (*angr.storage.file.SimFileDescriptor method*), 1615
concretize() (*angr.storage.file.SimFileDescriptorBase method*), 1615
concretize() (*angr.storage.file.SimFileDescriptorDuplex method*), 1616
concretize() (*angr.storage.file.SimPackets method*), 1613
concretize() (*in module angr.state_plugins.heap.utils*), 1561
concretize_2xm1() (*in module angr.engines.vex.heavy.concretizers*), 1075
concretize_abs_float64() (*in module angr.engines.vex.heavy.concretizers*), 1075
concretize_add32f04() (*in module angr.engines.vex.heavy.concretizers*), 1075
concretize_add64f02() (*in module angr.engines.vex.heavy.concretizers*), 1075
concretize_add_float64() (*in module angr.engines.vex.heavy.concretizers*), 1075
concretize_cmpf64() (*in module angr.engines.vex.heavy.concretizers*), 1075
concretize_div32f04() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_div64f02() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_divf64() (*in module angr.engines.vex.heavy.concretizers*), 1075
concretize_float32_to_float64() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_float64_to_float32() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_float64_to_int64s() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_fscale() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_fsqrt() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_fsqrt32() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_int32s_to_float64() (*in module*

*angr.engines.vex.heavy.concretizers*), 1076
concretize_int64s_to_float64() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_load_idx() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1637
concretize_mul32f04() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_mul64f02() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_mulf64() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_prem() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_prem_flags() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_read_addr() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 1633
concretize_reinterp_float64_as_int64() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_store_idx() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1637
concretize_sub32f04() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_sub64f02() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_trig_cos() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_trig_sin() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_trig_tan() (*in module angr.engines.vex.heavy.concretizers*), 1076
concretize_write_addr() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 1633
concretize_yl2x() (*in module angr.engines.vex.heavy.concretizers*), 1076
condition (*angr.procedures.posix.readdir.readdir attribute*), 1374
condition_str_to_function (*angr.engines.soot.expressions.condition.SimSootExpr_Condition attribute*), 1050
condition_str_to_function (*angr.engines.soot.expressions.SimSootExpr_Condition attribute*), 1048
condition_to_lambda() (*in module angr.exploration_techniques.common*), 1098
CONDITIONAL (*angr.state_plugins.sim_action.SimActionExit attribute*), 1585
CONDITIONAL (*angr.state_plugins.SimActionExit attribute*), 1504
ConditionalBreakLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 635
ConditionalBreakNode (*class in angr.analyses.decompiler.structurer_nodes*), 713
ConditionalJump (*class in angr.ailment.statement*), 224
ConditionalMixin (*class in angr.storage.memory_mixins.conditional_store_mixin*), 1634
ConditionalRegion (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl*), 639
ConditionConstantPropagation (*class in angr.analyses.decompiler.optimization_passes.condition_constpro*), 562
ConditionLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 635
ConditionNode (*class in angr.analyses.decompiler.structurer_nodes*), 711
ConditionProcessor (*class in angr.analyses.decompiler.condition_processor*), 408
confidence (*angr.analyses.language_detector.LanguageDetectionResult attribute*), 786
confidence (*angr.analyses.language_detector.LanguageDetector property*), 787
configure_project() (*angr.simos.simos.SimOS method*), 1488
CongruencyCheck (*class in angr.analyses.congruency_check*), 354
Const (*class in angr.ailment.expression*), 219
const_args_of_call() (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern method*), 532
CONST_TYPES (*angr.engines.light.ArithmeticExpression attribute*), 1006
CONST_TYPES (*angr.engines.light.data.ArithmeticExpression attribute*), 1010
const_val (*angr.knowledge_plugins.key_definitions.definition.DefinitionM attribute*), 1246
Constant (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 359
Constant (*angr.analyses.data_dep.DepNodeTypes attribute*), 356
CONSTANT (*angr.analyses.reaching_definitions.AtomKind attribute*), 806
CONSTANT (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 1241
Constant (*class in angr.analyses.stack_pointer_tracker*), 893
constant_after() (*angr.analyses.stack_pointer_tracker.StackPointerTrac*

*method*), 896
constant_after_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
constant_before() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
constant_before_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
constant_jump_targets (*angr.engines.pcode.lifter.IRSB property*), 1040
constant_jump_targets_and_jumpkinds (*angr.engines.pcode.lifter.IRSB property*), 1040
CONSTANT_PROPAGATION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
constant_value (*angr.analyses.purity.engine.DataSource attribute*), 801
ConstantChange (*class in angr.analyses.bindiff*), 280
ConstantCollector (*class in angr.analyses.decompiler.decompiler*), 414
ConstantCondition (*class in angr.analyses.decompiler.optimization_passes.constant_deref*), 562
ConstantDepNode (*class in angr.analyses.data_dep.dep_nodes*), 359
ConstantDereferences (*class in angr.analyses.decompiler.peephole_optimizations.constant_derefs*), 610
ConstantDereferencesSimplifier (*class in angr.analyses.decompiler.optimization_passes.const_derefs*), 563
ConstantOperand (*class in angr.analyses.disassembly*), 742
ConstantPropagation (*class in angr.analyses.binary_optimizer*), 279
ConstantResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.const_resolver*), 332
constants (*angr.engines.pcode.lifter.IRSB property*), 1040
ConstantSrc (*class in angr.knowledge_plugins.key_definitions.atoms*), 1244
ConstantValueManager (*class in angr.analyses.cfg.indirect_jump_resolvers.constant_value_manager*), 333
ConstantVariable (*class in angr.protos.variables_pb2*), 1405
ConstAssignmentInfo (*class in angr.analyses.decompiler.optimization_passes.inlined_memset_bind*), 577
ConstFormatEntry (*class in angr.protos.codegen_pb2*), 1401
ConstFormatEntry.FmtEntry (*class in angr.protos.codegen_pb2*), 1401
ConstFormatTsSerializer (*class in angr.angrdb.serializers.structured_code*), 954
ConstNode (*class in angr.analyses.loop_analysis.loop_analysis*), 788
ConstPropOptReverter (*class in angr.analyses.decompiler.optimization_passes.const_prop_reverter*), 563
constrain_all_zero() (*angr.analyses.identifier.identify.Identifier static method*), 785
constrain_callee_prototypes() (*angr.analyses.decompiler.clinic.Clinic method*), 408
constraint_hook() (*in module angr.state_plugins.trace_additions*), 1597
constraint_to_si() (*in module angr.utils.balancer*), 1674
ConstraintGraphNode (*class in angr.analyses.typehoon.simple_solver*), 901
ConstraintGraphTag (*class in angr.analyses.typehoon.simple_solver*), 901
CONSTRAINTS (*angr.state_plugins.inspect.EventType attribute*), 1565
constraints (*angr.state_plugins.solver.SimSolver property*), 1590
constraints_since() (*angr.state_plugins.history.SimStateHistory method*), 1562
ConstRetValueCollector (*class in angr.rust.analyses.rust_calling_convention.fact_collector*), 1408
construct() (*angr.knowledge_plugins.key_definitions.definition.Definition static method*), 1246
consumed_by_block (*angr.analyses.decompiler.known_patterns.finder.Kn attribute*), 507
consumed_by_block (*angr.analyses.decompiler.known_patterns.KnownPa attribute*), 461
consumed_by_block (*angr.analyses.decompiler.KnownPatternMatch attribute*), 383
consumed_stmt_idxs (*angr.analyses.decompiler.known_patterns.dsl.Mat attribute*), 490
ContainerNode (*class in angr.utils.graph*), 1684
contains() (*angr.knowledge_plugins.debug_variables.DebugVariable method*), 1192
contains_addr() (*angr.knowledge_plugins.functions.function_manager.F method*), 1222
contains_atom() (*angr.analyses.reaching_definitions.dep_graph.DepGra method*), 1826
content (*angr.analyses.reaching_definitions.subject.Subject*

*property*), 870
content (*angr.analyses.reassembler.Data property*), 874
content_gen() (*angr.storage.memory_mixins.paged_memory.pages.mar_list_page.MVListPage method*), 1651
context_sensitivity_level (*angr.analyses.cfg.cfg_base.CFGBase property*), 308
context_sensitivity_level (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 314
continue_addr (*angr.analyses.decompiler.structurer_nodes.LoopNode property*), 712
ContinueNode (*class in angr.analyses.decompiler.structurer_nodes*), 713
ContinueScanningNotification, 315
CONTRAVARIANT (*angr.analyses.typehoon.variance.Variance attribute*), 921
ControlFlowStructureCounter (*class in angr.analyses.decompiler.counters.seq_cf_structure_counter*), 412
ConvASub0ShrAnd (*class in angr.analyses.decompiler.peephole_optimizations.conv_a_sub0_shr_and*), 610
ConvenientMappingsMixin (*class in angr.storage.memory_mixins.convenient_mappings_mixin*), 1634
Convert (*class in angr.ailment.expression*), 219
convert() (*angr.ailment.converter_common.Converter static method*), 217
convert() (*angr.ailment.converter_pcode.PCodeIRSBConverter static method*), 217
convert() (*angr.ailment.converter_vex.VEXIRSBConverter static method*), 217
convert() (*angr.ailment.IRSBConverter static method*), 212
convert() (*angr.ailment.VEXIRSBConverter static method*), 214
convert_claripy_bool_ast() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 409
convert_claripy_bool_ast_core() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 409
convert_cppproto_to_py() (*in module angr.utils.library*), 1691
convert_cproto_to_py() (*in module angr.utils.library*), 1691
convert_from_lift() (*angr.ailment.converter_vex.VEXIRSBConverter static method*), 217
convert_from_lift() (*angr.ailment.VEXIRSBConverter static method*), 214
convert_prot() (*in module angr.procedures.win32.VirtualAlloc*), 1383
convert_variable_list() (*angr.knowledge_plugins.variables.variable_manager.VariableMa static method*), 1289
Converter (*class in angr.ailment.converter_common*), 216
ConvertTo (*class in angr.analyses.typehoon.typevars*), 920
ConvShlShr (*class in angr.analyses.decompiler.peephole_optimizations.con* 610
CooperationBase (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 1647
copy() (*angr.ailment.Block method*), 210
copy() (*angr.ailment.block.Block method*), 214
copy() (*angr.analyses.calling_convention.fact_collector.FactCollectorState method*), 288
copy() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
copy() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 312
copy() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 325
copy() (*angr.analyses.ddg.LiveDefinitions method*), 362
copy() (*angr.analyses.decompiler.optimization_passes.engine_base.Simpli method*), 574
copy() (*angr.analyses.decompiler.region_simplifiers.expr_folding.Statemen method*), 634
copy() (*angr.analyses.decompiler.ssailification.rewriting_state.RewritingS method*), 654
copy() (*angr.analyses.decompiler.ssailification.traversal_state.TraversalSta method*), 659
copy() (*angr.analyses.decompiler.structurer_nodes.CodeNode method*), 711
copy() (*angr.analyses.decompiler.structurer_nodes.LoopNode method*), 712
copy() (*angr.analyses.decompiler.structurer_nodes.MultiNode method*), 710
copy() (*angr.analyses.decompiler.structurer_nodes.SequenceNode method*), 711
copy() (*angr.analyses.fcp.fcp.FCPState method*), 745
copy() (*angr.analyses.purity.engine.StateType method*), 802
copy() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 835
copy() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsSt method*), 862
copy() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 895
copy() (*angr.analyses.typehoon.typevars.TypeVariables method*), 918
copy() (*angr.analyses.variable_recovery.variable_recovery.VariableRecov method*), 925

copy() (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery method*), 930
copy() (*angr.analyses.vfg.VFG method*), 936
copy() (*angr.concretization_strategies.base.SimConcretizationStrategy method*), 991
copy() (*angr.engines.pcode.lifter.IRSB method*), 1039
copy() (*angr.keyed_region.KeyedRegion method*), 1116
copy() (*angr.keyed_region.RegionObject method*), 1116
copy() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 1151
copy() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 1165
copy() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1167
copy() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 1175
copy() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 1174
copy() (*angr.knowledge_plugins.cfg.memory_data.MemoryData method*), 1178
copy() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1186
copy() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method*), 1180
copy() (*angr.knowledge_plugins.cfg.spilling_digraph.SpillingAdjDict method*), 1188
copy() (*angr.knowledge_plugins.custom_strings.CustomStrings method*), 1191
copy() (*angr.knowledge_plugins.data.Data method*), 1191
copy() (*angr.knowledge_plugins.functions.function.Function method*), 1218
copy() (*angr.knowledge_plugins.functions.function.FunctionInfo method*), 1209
copy() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1221
copy() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict method*), 1220
copy() (*angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager method*), 1249
copy() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1250
copy() (*angr.knowledge_plugins.key_definitions.liveness.Liveness method*), 1258
copy() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 1260
copy() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1263
copy() (*angr.knowledge_plugins.labels.Labels method*), 1264
copy() (*angr.knowledge_plugins.obfuscations.Obfuscations method*), 1264
copy() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
copy() (*angr.knowledge_plugins.plugin.KnowledgeBasePlugin method*), 1266
copy() (*angr.knowledge_plugins.propagations.propagation_manager.PropagationManager method*), 1271
copy() (*angr.knowledge_plugins.propagations.states.PropagatorState method*), 1274
copy() (*angr.knowledge_plugins.propagations.states.PropagatorVEXState method*), 1275
copy() (*angr.knowledge_plugins.structured_code.StructuredCodeManager method*), 1279
copy() (*angr.knowledge_plugins.types.TypesStore method*), 1280
copy() (*angr.knowledge_plugins.variables.variable_manager.DecompilationVariableManager method*), 1289
copy() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1289
copy() (*angr.knowledge_plugins.xrefs.xref.XRef method*), 1292
copy() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
copy() (*angr.misc.plugins.PluginPreset method*), 1320
copy() (*angr.procedures.definitions.SimLibrary method*), 1325
copy() (*angr.rust.knowledge_plugins.rust_calling_conventions.RustCallingConventionsAnalysis method*), 1413
copy() (*angr.rust.knowledge_plugins.type_hints.TypeHints method*), 1413
copy() (*angr.rust.sim_type.RustSimEnum method*), 1434
copy() (*angr.rust.sim_type.RustSimStruct method*), 1433
copy() (*angr.rust.sim_type.RustSimTypeArray method*), 1433
copy() (*angr.rust.sim_type.RustSimTypeFunction method*), 1432
copy() (*angr.rust.sim_type.RustSimTypeInt method*), 1431
copy() (*angr.rust.sim_type.RustSimTypeOption method*), 1435
copy() (*angr.rust.sim_type.RustSimTypeReference method*), 1432
copy() (*angr.rust.sim_type.RustSimTypeResult method*), 1435
copy() (*angr.rust.sim_type.RustSimTypeSize method*), 1431
copy() (*angr.rust.sim_type.RustSimTypeSlice method*), 1433
copy() (*angr.rust.sim_type.RustSimTypeStrRef method*), 1435
copy() (*angr.rust.sim_type.RustSimTypeUnit method*), 1435
copy() (*angr.rustylib.Segment method*), 1439
copy() (*angr.rustylib.SegmentList method*), 1439
copy() (*angr.sim_manager.SimulationManager method*), 1442
copy() (*angr.sim_state.SimState method*), 1455

copy() (*angr.sim_state_options.SimStateOptions method*), 1459
copy() (*angr.sim_type.SimCppClass method*), 1475
copy() (*angr.sim_type.SimCppClassValue method*), 1475
copy() (*angr.sim_type.SimStruct method*), 1471
copy() (*angr.sim_type.SimStructValue method*), 1472
copy() (*angr.sim_type.SimType method*), 1460
copy() (*angr.sim_type.SimTypeArray method*), 1468
copy() (*angr.sim_type.SimTypeBitfield method*), 1474
copy() (*angr.sim_type.SimTypeBool method*), 1465
copy() (*angr.sim_type.SimTypeBottom method*), 1462
copy() (*angr.sim_type.SimTypeChar method*), 1465
copy() (*angr.sim_type.SimTypeCppFunction method*), 1470
copy() (*angr.sim_type.SimTypeDouble method*), 1471
copy() (*angr.sim_type.SimTypeEnum method*), 1473
copy() (*angr.sim_type.SimTypeFd method*), 1466
copy() (*angr.sim_type.SimTypeFloat method*), 1470
copy() (*angr.sim_type.SimTypeFunction method*), 1469
copy() (*angr.sim_type.SimTypeInt method*), 1463
copy() (*angr.sim_type.SimTypeLength method*), 1470
copy() (*angr.sim_type.SimTypeNum method*), 1463
copy() (*angr.sim_type.SimTypeNumOffset method*), 1475
copy() (*angr.sim_type.SimTypePointer method*), 1467
copy() (*angr.sim_type.SimTypeReference method*), 1467
copy() (*angr.sim_type.SimTypeReg method*), 1462
copy() (*angr.sim_type.SimTypeString method*), 1468
copy() (*angr.sim_type.SimTypeTop method*), 1462
copy() (*angr.sim_type.SimTypeWideChar method*), 1465
copy() (*angr.sim_type.SimTypeWString method*), 1469
copy() (*angr.sim_type.SimUnion method*), 1472
copy() (*angr.sim_type.SimUnionValue method*), 1472
copy() (*angr.sim_type.TypeRef method*), 1461
copy() (*angr.sim_variable.SimComboRegisterVariable method*), 1479
copy() (*angr.sim_variable.SimConstantVariable method*), 1478
copy() (*angr.sim_variable.SimMemoryVariable method*), 1479
copy() (*angr.sim_variable.SimRegisterVariable method*), 1478
copy() (*angr.sim_variable.SimStackVariable method*), 1479
copy() (*angr.sim_variable.SimTemporaryVariable method*), 1478
copy() (*angr.sim_variable.SimVariable method*), 1478
copy() (*angr.state_plugins.edge_hitmap.SimStateEdgeHitmap method*), 1547
copy() (*angr.state_plugins.plugin.SimStatePlugin method*), 1578
copy() (*angr.state_plugins.sim_action.SimAction method*), 1584
copy() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1586
copy() (*angr.state_plugins.trace_additions.FormatInfo method*), 1596
copy() (*angr.state_plugins.trace_additions.FormatInfoDontConstrain method*), 1597
copy() (*angr.state_plugins.trace_additions.FormatInfoIntToStr method*), 1596
copy() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 1596
copy() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitm method*), 1654
copy() (*angr.storage.memory_mixins.regioned_memory.region_data.Regio method*), 1661
copy() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixi method*), 1662
copy() (*angr.utils.ins_addr_list.InsAddrList method*), 1690
copy_contents() (*angr.storage.memory_mixins.memory_mixin.MemoryM method*), 1640
copy_graph() (*angr.analyses.decompiler.clinic.Clinic method*), 407
copy_graph() (*in module angr.analyses.decompiler.utils*), 724
copy_graph_and_nodes() (*in module angr.analyses.decompiler.optimization_passes.duplication_revert* 573
copy_hooks() (*angr.misc.hookset.HookSet static method*), 1315
copy_to() (*angr.misc.hookset.HookedMethod method*), 1316
copy_without_tail() (*angr.engines.ail.callstack.AILCallStack method*), 1003
copy_without_tail() (*angr.state_plugins.callstack.CallStack method*), 1542
CORRECT_amd64g_dirtyhelper_CPUID_avx_and_cx16() (*in module angr.engines.vex.heavy.dirty*), 1077
correct_jump_targets() (*in module angr.analyses.decompiler.optimization_passes.duplication_revert* 573
CORRECT_x86g_dirtyhelper_CPUID_sse2() (*in module angr.engines.vex.heavy.dirty*), 1077
corroborated_by() (*in module angr.analyses.decompiler.known_patterns.gating*), 514
corroborated_by_pattern() (*in module angr.analyses.decompiler.known_patterns.gating*), 514
CorroboratedBy (*class in angr.analyses.decompiler.known_patterns.gating*), 512
CorroboratedByPattern (*class in*

*angr.analyses.decompiler.known_patterns.gating*), 514
CouldNotResolveException, 895
count() (*angr.exploration_techniques.spiller.PickledStatesDict method*), 1105
count() (*angr.state_plugins.history.TreeIter method*), 1563
count() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1650
count() (*angr.utils.ins_addr_list.InsAddrList method*), 1690
COVARIANT (*angr.analyses.typehoon.variance.Variance attribute*), 921
CppRef (*class in angr.analyses.decompiler.known_patterns.patterns*), 519
cprotos2py() (*in module angr.utils.library*), 1692
crash_windup() (*angr.exploration_techniques.tracer.Tracer class method*), 1109
crazy_scanf (*class in angr.procedures.stubs.crazy_scanf*), 1378
crc16() (*in module angr.analyses.flirt.flirt_matcher*), 756
create_angr_type_from_json() (*in module angr.procedures.definitions.parse_win32json*), 1334
create_conditionless_graph() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph.AILMergeGraph method*), 569
create_extract_vtable() (*angr.analyses.vtable.VtableFinder method*), 937
create_file() (*in module angr.procedures.posix.fdopen*), 1371
create_jump_target_var() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 410
create_mapping_to_merge_graph() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph.AILMergeGraph method*), 569
create_merged_subgraph() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph.AILMergeGraph method*), 571
create_new_reference() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 1574
create_phi_statements() (*angr.analyses.decompiler.ssailification.rewriting.SSARewriting method*), 653
create_server() (*in module angr.mcp.server*), 1307
create_session() (*angr.mcp.session.SessionManager method*), 1308
create_subregion() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 626
CreateMutexA (*class in angr.procedures.win32.mutex*), 1385
CreateMutexEx (*class in angr.procedures.win32.mutex*), 1385
CreationFailed (*angr.knowledge_plugins.cfg.cfg_node.CFGENode property*), 1175
CRegister (*class in angr.analyses.decompiler.structured_codegen.c*), 682
CRegisterMsg (*class in angr.protos.codegen_pb2*), 1399
CReturn (*class in angr.analyses.decompiler.structured_codegen.c*), 677
CReturnMsg (*class in angr.protos.codegen_pb2*), 1399
CRITICAL (*angr.analyses.decompiler.notes.decompilation_note.DecompilationNoteLevel attribute*), 538
CRITICAL (*angr.analyses.decompiler.notes.DecompilationNoteLevel attribute*), 536
CrossJumpReverter (*class in angr.analyses.decompiler.optimization_passes.cross_jump_reverter*), 564
crypto_scopes() (*angr.analyses.deobfuscator.scope_ops_analyzer.ScopeOpsAnalyzer method*), 738
cs_operand (*angr.analyses.disassembly.Operand property*), 742
CStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 673
CStatements (*class in angr.analyses.decompiler.structured_codegen.c*), 674
CStatementsMsg (*class in angr.protos.codegen_pb2*), 1400
CStructField (*class in angr.analyses.decompiler.structured_codegen.c*), 678
CStructFieldMsg (*class in angr.protos.codegen_pb2*), 1400
CStructFieldNameDef (*class in angr.analyses.decompiler.structured_codegen.c*), 684
CStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.c*), 684
CStructuredCodeWalker (*class in angr.analyses.decompiler.structured_codegen.c*), 686
CSwitchCase (*class in angr.analyses.decompiler.structured_codegen.c*), 676
CSwitchCaseEntry (*class in angr.protos.codegen_pb2*), 1400
CSwitchCaseMsg (*class in angr.protos.codegen_pb2*), 1400
ctype2rust() (*angr.rust.typehoon.translator.RustTypeTranslator method*), 1436
CTYPE_B_LOC (*in module angr.analyses.decompiler.known_patterns.ctype_tables*),

489
CTYPE_PREDICATES (*in module angr.analyses.decompiler.known_patterns.ctype_tables*), 490
CTypeCast (*class in angr.analyses.decompiler.structured_codegen.c*), 681
CTypeCastMsg (*class in angr.protos.codegen_pb2*), 1400
CUnaryOp (*class in angr.analyses.decompiler.structured_codegen.c*), 680
CUnaryOpMsg (*class in angr.protos.codegen_pb2*), 1400
CUnsupportedStatement (*class in angr.analyses.decompiler.structured_codegen.c*), 678
CUnsupportedStatementMsg (*class in angr.protos.codegen_pb2*), 1400
cur_ptr (*angr.procedures.ntdll.exceptions.KiUserExceptionDispatcher attribute*), 1369
current_function_address (*angr.state_plugins.callstack.CallStack property*), 1542
current_function_address() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 834
current_return_target (*angr.state_plugins.callstack.CallStack property*), 1543
current_stack_pointer (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 326
current_stack_pointer (*angr.state_plugins.callstack.CallStack property*), 1543
current_vvar_id (*angr.analyses.decompiler.ssailification.rewriting_engine.SimEngineSSARewriting property*), 654
custom_string() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
CustomStrings (*class in angr.knowledge_plugins.custom_strings*), 1191
CuteFormatter (*class in angr.misc.loggers*), 1316
cvar (*angr.analyses.decompiler.edits.resolve.ResolvedVariable attribute*), 447
cvar (*angr.analyses.decompiler.edits.ResolvedVariable attribute*), 428
CVariable (*class in angr.analyses.decompiler.structured_codegen.c*), 679
CVariableField (*class in angr.analyses.decompiler.structured_codegen.c*), 680
CVariableFieldMsg (*class in angr.protos.codegen_pb2*), 1400
CVariableMsg (*class in angr.protos.codegen_pb2*), 1400
CVectorConvert (*class in angr.analyses.decompiler.structured_codegen.c*), 683
CVectorConvertMsg (*class in angr.protos.codegen_pb2*), 1400
CVEXCCallExpression (*class in angr.analyses.decompiler.structured_codegen.c*), 683
CVEXCCallExpressionMsg (*class in angr.protos.codegen_pb2*), 1400
CWhileLoop (*class in angr.analyses.decompiler.structured_codegen.c*), 674
cyan (*angr.misc.ansi.BackgroundColor attribute*), 1314
cyan (*angr.misc.ansi.Color attribute*), 1314
cyclomatic_complexity (*angr.knowledge_plugins.functions.function.Function property*), 1212

## D

Data (*class in angr.analyses.reassembler*), 874
Data (*class in angr.knowledge_plugins.data*), 1191
data_graph (*angr.analyses.ddg.DDG property*), 365
data_ptr() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
data_sub_graph() (*angr.analyses.ddg.DDG method*), 365
DataDependencyGraphAnalysis (*class in angr.analyses.data_dep.data_dependency_analysis*), 358
DataGraphError, 361
DataGraphMeta (*class in angr.analyses.datagraph_meta*), 361
DataLabel (*class in angr.analyses.reassembler*), 870
DataNormalizationMixin (*class in angr.storage.memory_mixins.bv_conversion_mixin*), 1634
DATAREF_HINTS (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 319
DataRefDesc (*class in angr.analyses.decompiler.clinic*), 403
DataSource (*class in angr.analyses.purity.engine*), 801
DataTransformationEmbedder (*class in angr.analyses.deobfuscator.data_transformation_embedder*), 737
DataType (*in module angr.engines.ail.engine_light*), 1003
DataType_co (*in module angr.analyses.purity.engine*), 803
DataUsage (*class in angr.analyses.purity.engine*), 801
db_batch_size (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1186
db_batch_size (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNod property*), 1180
db_compatible() (*angr.angrdb.db.AngrDB method*), 941
DbBookmark (*class in angr.angrdb.models*), 947

DbCallGraph (*class in angr.angrdb.models*), 944
DbCFGModel (*class in angr.angrdb.models*), 943
DbComment (*class in angr.angrdb.models*), 946
DbDecompilationCache (*class in angr.angrdb.models*), 946
DbDecVariableCollection (*class in angr.angrdb.models*), 945
DbFunction (*class in angr.angrdb.models*), 944
dbg_comments() (*angr.analyses.reassembler.Instruction method*), 872
dbg_draw() (*angr.knowledge_plugins.functions.function.Function method*), 1216
dbg_draw() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1224
dbg_print() (*angr.knowledge_plugins.functions.function.Function method*), 1216
dbg_print() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionMapMixin method*), 1662
dbg_print_irsb() (*angr.annocfg.AnnotatedCFG method*), 958
dbg_print_stack() (*angr.sim_state.SimState method*), 1455
dbg_repr() (*angr.ailment.Block method*), 210
dbg_repr() (*angr.ailment.block.Block method*), 214
dbg_repr() (*angr.analyses.backward_slice.BackwardSlice method*), 278
dbg_repr() (*angr.analyses.cfg.cfb.CFBlanket method*), 306
dbg_repr() (*angr.analyses.ddg.DDG method*), 365
dbg_repr() (*angr.analyses.decompiler.clinic.Clinic method*), 407
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.BaseNode method*), 711
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.BreakNode method*), 713
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.ConditionNode method*), 712
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.ConditionalBreakNode method*), 713
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.ConditionalJumpNode method*), 712
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.ContinueNode method*), 713
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.LoopNode method*), 712
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.MultiNode method*), 710
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.SequenceNode method*), 711
dbg_repr() (*angr.analyses.decompiler.structurer_nodes.SwitchCaseNode method*), 713
dbg_repr() (*angr.annocfg.AnnotatedCFG method*), 958
dbg_repr() (*angr.blade.Blade method*), 959
dbg_repr() (*angr.keyed_region.KeyedRegion method*), 1117
dbg_repr() (*angr.rust.structuring.structurer_nodes.IfLetNode method*), 1436
dbg_repr() (*angr.rust.structuring.structurer_nodes.PatternMatchNode method*), 1436
dbg_repr() (*angr.state_plugins.callstack.CallStack method*), 1544
dbg_repr_run() (*angr.analyses.backward_slice.BackwardSlice method*), 278
DbInformation (*class in angr.angrdb.models*), 942
DbKnowledgeBaseMetadata (*class in angr.angrdb.models*), 943
DbLabel (*class in angr.angrdb.models*), 947
DbObject (*class in angr.angrdb.models*), 942
DbStructuredCode (*class in angr.angrdb.models*), 945
DbVariableCollection (*class in angr.angrdb.models*), 944
DbXRefs (*class in angr.angrdb.models*), 946
DDG (*class in angr.analyses.ddg*), 364
DDGJob (*class in angr.analyses.ddg*), 362
DDGView (*class in angr.analyses.ddg*), 364
DDGViewInstruction (*class in angr.analyses.ddg*), 364
DDGViewItem (*class in angr.analyses.ddg*), 363
deactivate() (*angr.misc.plugins.PluginPreset method*), 1319
dead_ref() (*angr.state_hierarchy.StateHierarchy method*), 1494
dead_vvar_ids (*angr.analyses.s_propagator.SPropagator property*), 879
DeadAssignment (*class in angr.analyses.binary_optimizer*), 279
DeadblockRemover (*class in angr.analyses.decompiler.optimization_passes.deadblock_remover*), 565
deadends (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 314
deallocate (*class in angr.procedures.cgc.deallocate*), 1323
DEBUG (*angr.analyses.decompiler.notes.decompilation_note.DecompilationNoteLevel attribute*), 538
DEBUG (*angr.analyses.decompiler.notes.DecompilationNoteLevel attribute*), 536
debug() (*angr.sim_manager.ErrorRecord method*), 1448
DebugVariable (*class in angr.knowledge_plugins.debug_variables*), 1192
DebugVariableContainer (*class in angr.knowledge_plugins.debug_variables*), 1192
DebugVariableManager (*class in angr.knowledge_plugins.debug_variables*), 1192
dec_active_workers() (*angr.distributed.server.Server*

*method*), 994
decode_instruction() (*in module angr.analyses.disassembly_utils*), 743
decode_utf16_string() (*in module angr.utils.strings*), 1699
DecodingAssumption (*class in angr.analyses.cfg.cfg_fast*), 315
DecompilationCache (*class in angr.analyses.decompiler.decompilation_cache*), 412
DecompilationCache (*class in angr.protos.decompilation_cache_pb2*), 1402
DecompilationCache.NotesJsonEntry (*class in angr.protos.decompilation_cache_pb2*), 1402
DecompilationComplexityError, 1301
DecompilationEditError, 437
DecompilationError, 1301
DecompilationNote (*class in angr.analyses.decompiler.notes.decompilation_note*), 538
DecompilationNoteLevel (*class in angr.analyses.decompiler.notes.decompilation_note*), 538
DecompilationOption (*class in angr.analyses.decompiler.decompilation_options*), 413
DecompilationParameters (*class in angr.protos.decompilation_cache_pb2*), 1402
DecompilationParameters.ExprCommentsEntry (*class in angr.protos.decompilation_cache_pb2*), 1402
DecompilationParameters.StmtCommentsEntry (*class in angr.protos.decompilation_cache_pb2*), 1402
DecompilationPreset (*class in angr.analyses.decompiler.presets.preset*), 622
DecompilationVariableManager (*class in angr.knowledge_plugins.variables.variable_manager*), 1289
DECOMPILE (*angr.analyses.decompiler.clinic.ClinicMode attribute*), 403
DECOMPILE (*angr.analyses.decompiler.ClinicMode attribute*), 373
decompile_function() (*in module angr.mcp.server*), 1304
decompile_functions() (*in module angr.analyses.decompiler.utils*), 725
Decompiler (*class in angr.analyses.decompiler.decompiler*), 414
deconvert_prot() (*in module angr.procedures.win32.VirtualAlloc*), 1383
deep_copy() (*angr.ailment.Block method*), 210
deep_copy() (*angr.ailment.block.Block method*), 214
deepcopy_ail_anyjump() (*in module angr.analyses.decompiler.optimization_passes.duplication_revert*), 573
deepcopy_ail_condjump() (*in module angr.analyses.decompiler.optimization_passes.duplication_revert*), 573
deepcopy_ail_jump() (*in module angr.analyses.decompiler.optimization_passes.duplication_revert*), 573
Def (*in module angr.analyses.decompiler.ssailification.ssailification*), 655
DEF_EQ_SAME_BLOCK (*angr.analyses.decompiler.ail_simplifier.DefEqRelati attribute*), 397
def_fn (*angr.analyses.decompiler.known_patterns.dsl.MatchCtx attribute*), 491
DEF_IN_EQ_PRED_BLOCK (*angr.analyses.decompiler.ail_simplifier.DefEqRelation attribute*), 397
DEF_IS_FUNCARG (*angr.analyses.decompiler.ail_simplifier.DefEqRelation attribute*), 397
def_order (*angr.sim_type.SimStruct property*), 1471
DEFAULT (*angr.state_plugins.sim_action.SimActionExit attribute*), 1585
DEFAULT (*angr.state_plugins.SimActionExit attribute*), 1504
default_cc() (*in module angr.calling_conventions*), 983
default_cc_for_project() (*in module angr.calling_conventions*), 984
default_enabled (*angr.analyses.decompiler.known_patterns.KnownPatte attribute*), 458
default_enabled (*angr.analyses.decompiler.known_patterns.KnownPatte attribute*), 463
default_enabled (*angr.analyses.decompiler.known_patterns.pattern.Kno attribute*), 521
default_enabled (*angr.analyses.decompiler.known_patterns.templates.K attribute*), 532
default_enabled (*angr.analyses.decompiler.KnownPattern attribute*), 379
default_engine (*angr.factory.AngrObjectFactory property*), 1110
default_indirect_jump_resolvers() (*in module angr.analyses.cfg.indirect_jump_resolvers.default_resolvers*), 333
default_simtype_from_bits() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCod method*), 685
default_simtype_from_size() (*angr.analyses.decompiler.structured_codegen.rust.RustStructure method*), 708
DefaultChainMapCOW (*class in angr.utils.cowdict*), 1676
DefaultFillerMixin (*class in angr.storage.memory_mixins.default_filler_mixin*),

1635
DefaultListPagesMemory (*class in angr.storage.memory_mixins*), 1619
DefaultMemory (*class in angr.storage.memory_mixins*), 1619
DefEqRelation (*class in angr.analyses.decompiler.ail_simplifier*), 397
defer_cleanup() (*angr.state_hierarchy.StateHierarchy method*), 1494
DefInfo (*class in angr.analyses.decompiler.ssailification.traversal_engine*), 655
Definition (*class in angr.knowledge_plugins.key_definitions.definitions*), 1246
DefinitionAnnotation (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 1249
DefinitionMatchPredicate (*class in angr.knowledge_plugins.key_definitions.definitions*), 1245
definitions (*angr.analyses.ddg.DDGViewInstruction property*), 364
DefUseChain (*class in angr.analyses.vsa_ddg*), 936
delete() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
delete() (*angr.state_plugins.filesystem.SimMount method*), 1550
delete_reference() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 1574
delete_uc() (*angr.state_plugins.unicorn_engine.Unicorn static method*), 1603
DeleteGlobalRef (*class in angr.procedures.java_jni.global_and_local_refs*), 1341
DeleteLocalRef (*class in angr.procedures.java_jni.global_and_local_refs*), 1342
demangle() (*in module angr.rust.utils.demangler*), 1438
demangled_name (*angr.knowledge_plugins.functions.function.Function property*), 1217
demote() (*angr.state_plugins.history.SimStateHistory method*), 1562
DeobfuscatedString (*class in angr.analyses.decompiler.notes.deobfuscated_strings*), 539
DeobfuscatedStringsNote (*class in angr.analyses.decompiler.notes.deobfuscated_strings*), 539
dep_graph (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
dep_graph (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 868
dependents (*angr.analyses.ddg.DDGViewItem property*), 364
depends() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 840
depends_on (*angr.analyses.ddg.DDGViewItem property*), 364
DepGraph (*class in angr.analyses.reaching_definitions.dep_graph*), 835
DephicationBase (*class in angr.analyses.decompiler.dephication.dephication_base*), 418
DepNodeTypes (*class in angr.analyses.data_dep.dep_nodes*), 359
deprecated() (*in module angr.misc.ux*), 1322
depth (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseHeadStatement property*), 714
depth (*angr.state_plugins.sim_action_object.SimActionObject property*), 1586
deref (*angr.state_plugins.debug_variables.SimDebugVariable property*), 1546
deref (*angr.state_plugins.view.SimMemView property*), 1607
deref() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 866
deref() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1257
deref_vvar_and_offset() (*in module angr.rust.utils.ail*), 1438
DerefCoercionSimplifier (*class in angr.rust.optimization_passes.deref_coercion_simplifier*), 1422
DerefCoercionSimplifierUninlined (*class in angr.rust.optimization_passes.deref_coercion_simplifier_uninlined*), 1422
dereference_simtype() (*in module angr.utils.types*), 1701
dereference_simtype_by_lib() (*in module angr.utils.types*), 1702
DerefSize (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 1249
DerivedTypeVariable (*class in angr.analyses.typehoon.typevars*), 917
describe_variables() (*angr.state_plugins.solver.SimSolver method*), 1588
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier attribute*), 560
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.BasePointerSaveSimplifier attribute*), 540
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.call_stmt_rewriter.CallStatementRewriter attribute*), 560
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.CallStatementRewriter attribute*), 541
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.code_motion.CodeMotionOptimizationPass attribute*), 561
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.CodeMotionO

*attribute*), 541
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.cross_jump_object_merging.CrossJumpObjectMerging attribute*), 562
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.duplication_reverter.DuplicationReverter attribute*), 542
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILEngine attribute*), 563
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.OpSwapper attribute*), 564
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.flip_boolean_cmp.FlipBooleanCmp attribute*), 543
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.inlined_strcpy.InlinedStrcpy attribute*), 543
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.inlined_strlen.InlinedStrlen attribute*), 564
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy.InlinedWcscpy attribute*), 543
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 565
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ite_region_converter.ITERegionConverter attribute*), 544
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 565
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 566
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.multi_simplifier_pass.MultiSimplifier attribute*), 544
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.multiple_def_ret_val_simplifier.MultipleDefRetValSimplifier attribute*), 570
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.return_duplicator_base.CodeDuplicator attribute*), 566
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.CodeDuplicator attribute*), 545
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 574
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanaryInstruction attribute*), 574
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.string_dereference.StringDereferenceSimplification attribute*), 546
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier.WinStackCanarySimplifier attribute*), 576
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcSimplifier attribute*), 547
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcInstruction attribute*), 577
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcExpression attribute*), 547
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcAssignment attribute*), 577
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcStatement attribute*), 578
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcSimplifier attribute*), 578
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcInstruction attribute*), 581
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcExpression attribute*), 581
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcAssignment attribute*), 582
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcStatement attribute*), 548
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GetPcSimplifier attribute*), 548
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_overflow_simplifier.X86OverflowSimplifier attribute*), 549
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_overflow_simplifier.OverflowInstruction attribute*), 549
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_overflow_simplifier.OverflowExpression attribute*), 550
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_overflow_simplifier.OverflowAssignment attribute*), 550
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_overflow_simplifier.OverflowStatement attribute*), 583
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.x86_overflow_simplifier.X86OverflowSimplifier attribute*), 583
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.xor_simplifier.XorSimplifier attribute*), 547
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.xor_simplifier.XorInstruction attribute*), 548
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.known_pattern_simplifier.KnownPatternSimplifier attribute*), 583
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.known_pattern_simplifier.KnownPatternInstruction attribute*), 550
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 585
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.LoweredSwitchSimplifier attribute*), 551
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.mips_gp_setting_optimizer.MipsGpSettingOptimizer attribute*), 585
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.MipsGpSettingOptimizer attribute*), 552
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 586
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 552
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 587
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.peephole_simplifier_multi_instr.PeepholeSimplifierMultiInstruction attribute*), 590
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.register_save_simplifier.RegisterSaveSimplifier attribute*), 590
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.register_save_simplifier.InlinedMemcpySimplifier attribute*), 591
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.RegisterSaveSimplifier attribute*), 553
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.simplify_function_epilogues.FunctionEpilogueSimplifier attribute*), 553
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier*

*attribute*), 591
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator*
*attribute*), 591
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.RetAddrSimplifier*
*attribute*), 554
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.ret_dup*
*attribute*), 593
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.symbol_deref*
*attribute*), 594
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.switch_reuse_edge*
*attribute*), 554
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.switch_reuse_edge_simplifier*
*attribute*), 554
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.tail_call_duplicator_low*
*attribute*), 555
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.tail_call_duplicator_simplifier*
*attribute*), 594
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.StaticVVarRewriter*
*attribute*), 596
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.StaticVVarRewriter*
*attribute*), 556
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator*
*attribute*), 596
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.switch_reused_entry_rewriter*
*attribute*), 597
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator*
*attribute*), 557
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryRewriter*
*attribute*), 557
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.tag_slicer.TagSlicer*
*attribute*), 597
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.TagSlicer*
*attribute*), 558
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier*
*attribute*), 598
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier*
*attribute*), 558
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.X86gcc_cdqcx_simplifier.X86GccCdqCxSimplifier*
*attribute*), 598
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.X86GccCdqCxSimplifier*
*attribute*), 558
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBaseHelper*
*attribute*), 604
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase*
*attribute*), 603
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase*
*attribute*), 602
DESCRIPTION (*angr.analyses.decompiler.peephole_optimizations.evaluate_constants.EvaluateConstants*
*attribute*), 611
DESCRIPTION (*angr.analyses.deobfuscator.string_obf_opt_pass.StringObfType3Rewriter*
*attribute*), 739
DESCRIPTION (*angr.analyses.deobfuscator.StringObfType3Rewriter*
*attribute*), 733
DESCRIPTION (*angr.analyses.decompiler.optimization_passes.pre_pattern_match_simplifier*
*attribute*), 1427
description (*angr.state_plugins.history.SimStateHistory*
*property*), 1562
description() (*angr.analyses.decompiler.region_identifier.RegionIdentifier.HighUnicorn*
*method*), 1604
description() (*angr.analyses.decompiler.structurer.base.BaseStructurer.Data*
*method*), 874
description() (*angr.analyses.decompiler.region_overlay.RegionOverlay*
*method*), 629
description() (*angr.analyses.decompiler.region_overlay.RegionOverlayHigh*
*method*), 628
Detail (*angr.analyses.decompiler.knowledge_plugins.propagations.prop_value*), 1268
detail() (*angr.simplifiers.StackCanarySimplifier.ResolvedVariable*
*method*), 447
detect_cxx_rtti() (*in module angr.analyses.decompiler.known_patterns.context*), 488
detect_linux_kernel_object() (*in module angr.analyses.decompiler.known_patterns.context*), 487
detect_default_case_duplicate() (*in module angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator*
*method*), 1322
detect_windows_kernel_driver() (*in module angr.analyses.decompiler.known_patterns.context*), 488
determine() (*angr.analyses.typehoon.simple_solver.SimpleSolver*
*method*), 998
determine_smc() (*angr.analyses.smc.TraceClassifier*
*method*), 892
DetermineLoadSizes (*class in angr.analyses.decompiler.optimization_passes.determine_load_sizes*), 565
df() (*angr.utils.dominance.DominanceWithStackCanarySimplifier*
*method*), 1677
DFAConstraintSolver (*class in angr.analyses.typehoon.dfa*), 898
DFA64.xic (*class in angr.analyses.X86Gcc.GccPrintf*), 1419
DFS (*class in angr.exploration_techniques.dfs*), 1099
dfs_back_edges() (*in module angr.utils.graph*), 1683
dfs_postorder_nodes_deterministic() (*angr.utils.graph.DirectEdgeBaseHelper*
*method*), 1688
dfs_postorder_nodes_deterministic() (*angr.utils.graph.GraphUtils static method*), 1686
dfs_postorder_nodes_deterministic_multi() (*angr.utils.graph.GraphUtils static method*), 1686
diagnose() (*angr.analyses.pathfinder.Pathfinder*
*method*), 795
dirty_strkey_to_intkey()

(*angr.angrdb.serializers.structured_code.StructuredCodeSerializer static method*), 955
DictBackerMixin (*class in angr.storage.memory_mixins.paged_memory.page_backer_mixins*), 1641
Difference (*class in angr.analyses.bindiff*), 280
difference() (*angr.sim_state_options.SimStateOptions method*), 1458
differing_blocks (*angr.analyses.bindiff.BinDiff property*), 282
differing_blocks (*angr.analyses.bindiff.FunctionDiff property*), 281
differing_constants() (*in module angr.analyses.bindiff*), 280
differing_functions (*angr.analyses.bindiff.BinDiff property*), 282
differing_functions_with_consts() (*angr.analyses.bindiff.BinDiff method*), 282
direct_next (*angr.engines.pcode.lifter.IRSB property*), 1040
DirectedGraphHelper (*class in angr.utils.graph*), 1688
Director (*class in angr.exploration_techniques.director*), 1101
Dirent (*in module angr.procedures.posix.readdir*), 1374
dirty (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
dirty (*angr.knowledge_plugins.functions.function.Function property*), 1214
DIRTY (*angr.state_plugins.inspect.EventType attribute*), 1565
dirty_args (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
dirty_args (*angr.state_plugins.InspectAttrs attribute*), 1499
dirty_func() (*in module angr.knowledge_plugins.functions.function*), 1209
dirty_handler (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
dirty_handler (*angr.state_plugins.InspectAttrs attribute*), 1499
dirty_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1013
dirty_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
dirty_name (*angr.state_plugins.InspectAttrs attribute*), 1499
dirty_result (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
dirty_result (*angr.state_plugins.InspectAttrs attribute*), 1499
DirtyAddrsMixin (*class in angr.storage.memory_mixins.dirty_addrs_mixin*), 1635
DirtyDiGraph (*class in angr.knowledge_plugins.cfg.spilling_digraph*), 1187
DirtyExpression (*class in angr.ailment.expression*), 216
DirtyRewriterBase (*class in angr.analyses.decompiler.dirty_rewriters.rewriter_base*), 421
DirtyStatement (*class in angr.ailment.statement*), 224
disable_profiling() (*in module angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 338
disable_root_logger() (*angr.misc.loggers.Loggers method*), 1316
disable_timing() (*in module angr.state_plugins.solver*), 1587
disassemble_block() (*angr.analyses.reassembler.Reassembler method*), 875
DisassemblerBlock (*class in angr.block*), 959
DisassemblerInsn (*class in angr.block*), 959
disassembly (*angr.block.Block property*), 961
disassembly (*angr.engines.pcode.lifter.IRSB property*), 1041
Disassembly (*class in angr.analyses.disassembly*), 742
disassembly_dirty (*angr.analyses.decompiler.edits.Refresh attribute*), 427
disassembly_dirty (*angr.analyses.decompiler.edits.results.Refresh attribute*), 449
DisassemblyPiece (*class in angr.analyses.disassembly*), 739
discard() (*angr.knowledge_plugins.structured_code.StructuredCodeMana method*), 1279
discard() (*angr.sim_state_options.SimStateOptions method*), 1458
discard_by_prefix() (*angr.knowledge_plugins.propagations.propagation_manager.Pro method*), 1271
discard_input() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
discard_output() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
discard_plugin_preset() (*angr.misc.plugins.PluginHub method*), 1318
dispatch() (*angr.procedures.ntdll.exceptions.KiUserExceptionDispatcher method*), 1369
display_name (*angr.analyses.data_dep.dep_nodes.VarDepNode property*), 360
display_name (*angr.knowledge_plugins.comments.CommentKind property*), 1189
dissect_instruction() (*angr.analyses.disassembly.Instruction method*), 741
dissect_instruction_by_default() (*angr.analyses.disassembly.Instruction*

*method*), 741
dissect_instruction_for_arm() (*angr.analyses.disassembly.Instruction method*), 741
dissolve() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 631
DivSimplifier (*class in angr.analyses.decompiler.optimization_passes.div_simplifier*), 566
DivSimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.div_simplifier*), 565
do_full_xrefs() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 325
do_it() (*in module angr.procedures.definitions.parse_win32json*), 1334
do_pretests() (*angr.analyses.identifier.functions.recv_until.receive_until_fd method*), 778
do_pretests() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 777
do_trace() (*angr.analyses.identifier.identify.Identifier method*), 785
dominance_frontiers (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase property*), 929
DominanceFrontier (*class in angr.analyses.dominance_frontier*), 743
dominates() (*angr.utils.doms.IncrementalDominators method*), 1678
dominates() (*in module angr.utils.graph*), 1683
dominates_by_intervals() (*in module angr.utils.graph*), 1684
Dominators (*class in angr.utils.graph*), 1685
done (*angr.analyses.vfg.AnalysisTask property*), 933
done (*angr.analyses.vfg.CallAnalysis property*), 933
done (*angr.analyses.vfg.FunctionAnalysis property*), 933
DoNotUpdate (*angr.analyses.calling_convention.calling_convention.SimCCUpdateArgumentsOption attribute*), 285
downsize() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 313
downsize() (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis method*), 764
downsize() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 866
downsize() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 929
downsize() (*angr.knowledge_plugins.cfg.cfg_node.CFGENode method*), 1175
downsize() (*angr.knowledge_plugins.propagations.propagation_model.PropagationModel method*), 1272
downsize() (*angr.sim_state.SimState method*), 1454
downsize() (*angr.state_plugins.inspect.SimInspector method*), 1572
downsize() (*angr.state_plugins.sim_action.SimAction method*), 1584
downsize() (*angr.state_plugins.solver.SimSolver method*), 1589
downsize_region() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 929
DreamStructurer (*class in angr.analyses.decompiler.structuring.dream*), 716
DrillerCore (*class in angr.exploration_techniques.driller_core*), 1101
DROP (*angr.sim_manager.SimulationManager attribute*), 1442
DROP (*angr.SimulationManager attribute*), 199
drop() (*angr.sim_manager.SimulationManager method*), 1446
drop_bad_functions() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 325
drop_bad_functions() (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method*), 325
drop_db() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method*), 1278
drop_edge_marks_from() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 629
dst (*angr.analyses.decompiler.known_patterns.dsl.PCallStmt attribute*), 504
dst (*angr.analyses.decompiler.known_patterns.PCallStmt attribute*), 468
DummyStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.dummy*), 691
dump() (*angr.angrdb.db.AngrDB method*), 941
dump() (*angr.angrdb.serializers.bookmarks.BookmarksSerializer static method*), 949
dump() (*angr.angrdb.serializers.callgraph.CallGraphSerializer static method*), 950
dump() (*angr.angrdb.serializers.cfg_model.CFGModelSerializer static method*), 951
dump() (*angr.angrdb.serializers.comments.CommentsSerializer static method*), 951
dump() (*angr.angrdb.serializers.funcs.FunctionManagerSerializer static method*), 951
dump() (*angr.angrdb.serializers.kb.KnowledgeBaseSerializer static method*), 952
dump() (*angr.angrdb.serializers.labels.LabelsSerializer static method*), 952
dump() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 954
dump() (*angr.angrdb.serializers.structured_code.StructuredCodeManagerS static method*), 955
dump() (*angr.angrdb.serializers.variables.VariableManagerSerializer*

*static method*), 955
dump() (*angr.angrdb.serializers.xrefs.XRefsSerializer static method*), 956
dump() (*angr.calling_conventions.AllocHelper method*), 963
dump() (*angr.tablespecs.StringTableSpec method*), 1668
dump_constraint_graph() (*angr.analyses.typehoon.simple_solver.SimpleSolver static method*), 905
dump_dvars() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 955
dump_file_by_path() (*angr.state_plugins.posix.SimSystemPosix method*), 1582
dump_graph() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer static method*), 718
dump_mapping() (*in module angr.procedures.definitions.parse_syscalls_from_local_system*), 1334
dumps() (*angr.state_plugins.posix.SimSystemPosix method*), 1582
dumps() (*angr.vaults.Vault method*), 1704
dup (*class in angr.procedures.posix.dup*), 1371
dup2 (*class in angr.procedures.posix.dup*), 1371
dup3 (*class in angr.procedures.posix.dup*), 1371
duplicated_stmt_idxs (*angr.analyses.decompiler.known_patterns.finder.KnownPatternMatch attribute*), 507
duplicated_stmt_idxs (*angr.analyses.decompiler.known_patterns.KnownPatternMatch attribute*), 461
duplicated_stmt_idxs (*angr.analyses.decompiler.KnownPatternMatch attribute*), 383
DUPLICATION_CHECK (*angr.analyses.decompiler.structured_codegen.base.PositionMapping attribute*), 668
DUPLICATION_CHECK (*angr.analyses.decompiler.structured_codegen.PositionMapping attribute*), 666
DuplicationReverter (*class in angr.analyses.decompiler.optimization_passes.duplication_reverter*), 570
DURING_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPassStage attribute*), 587
DURING_REGION_IDENTIFICATION (*angr.analyses.decompiler.optimization_passes.OptimizationPassStage attribute*), 552
dwarf_cfa (*angr.state_plugins.debug_variables.SimDebugVariablePlugin property*), 1546
dwarf_cfa_approx (*angr.state_plugins.debug_variables.SimDebugVariablePlugin property*), 1546
dwTime (*angr.procedures.win32.GetLastInputInfo.GetLastInputInfo attribute*), 1382
DYNAMIC_RET (*angr.procedures.libc.error.error attribute*), 1353
DYNAMIC_RET (*angr.sim_procedure.SimProcedure attribute*), 1450
DYNAMIC_RET (*angr.SimProcedure attribute*), 190
dynamic_returns() (*angr.sim_procedure.SimProcedure method*), 1451
DynamicDictList (*class in angr.utils.dynamic_dictlist*), 1678

## E

E2BIG (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
E2BIG (*angr.state_plugins.SimSystemPosix attribute*), 1536
EACCES (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EACCES (*angr.state_plugins.SimSystemPosix attribute*), 1537
EAGAIN (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EAGAIN (*angr.state_plugins.SimSystemPosix attribute*), 1536
EagerEvaluation (*class in angr.analyses.decompiler.peephole_optimizations.eager_eval*), 611
EagerStdStringConcatenationPass (*class in angr.analyses.decompiler.optimization_passes.eager_std_string_c*), 573
EagerStdStringEvalPass (*class in angr.analyses.decompiler.optimization_passes.eager_std_string_e*), 574
EBADF (*angr.state_plugins.cgc.SimStateCGC attribute*), 1544
EBADF (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EBADF (*angr.state_plugins.SimStateCGC attribute*), 1525
EBADF (*angr.state_plugins.SimSystemPosix attribute*), 1536
EBUSY (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EBUSY (*angr.state_plugins.SimSystemPosix attribute*), 1537
ECHILD (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ECHILD (*angr.state_plugins.SimSystemPosix attribute*), 1536
Edge (*class in angr.protos.primitives_pb2*), 1403
edge_marked() (*angr.analyses.decompiler.region_overlay.RegionOverlayC method*), 632
edges (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1185
EditHooks (*class in angr.analyses.decompiler.edits.hooks*), 438

EditResult (*class in angr.analyses.decompiler.edits.results*), 449
EDOM (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
EDOM (*angr.state_plugins.SimSystemPosix attribute*), 1537
EEXIST (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EEXIST (*angr.state_plugins.SimSystemPosix attribute*), 1537
EFAULT (*angr.state_plugins.cgc.SimStateCGC attribute*), 1544
EFAULT (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EFAULT (*angr.state_plugins.SimStateCGC attribute*), 1525
EFAULT (*angr.state_plugins.SimSystemPosix attribute*), 1537
EFBIG (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
EFBIG (*angr.state_plugins.SimSystemPosix attribute*), 1537
EffectiveSizeExtractor (*class in angr.analyses.decompiler.expression_narrower*), 451
EH4ScopeRecord (*class in angr.analyses.cfg.pe_msvc_eh_structs*), 344
EH4ScopeTable (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
EH4ScopeTable (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
EH4ScopeTable (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
EH4ScopeTable (*class in angr.analyses.cfg.pe_msvc_eh_structs*), 343
EH_FRAME_HINTS (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 319
EHFuncInfo (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
EHFuncInfo (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
EHFuncInfo (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
EHHandlerType (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
EHHandlerType (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
EHHandlerType (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
EHTryBlockMap (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
EHTryBlockMap (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
EHTryBlockMap (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
EHUnwindMapEntry (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
EHUnwindMapEntry (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
EHUnwindMapEntry (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
EINTR (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EINTR (*angr.state_plugins.SimSystemPosix attribute*), 1536
EINVAL (*angr.state_plugins.cgc.SimStateCGC attribute*), 1544
EINVAL (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EINVAL (*angr.state_plugins.SimStateCGC attribute*), 1525
EINVAL (*angr.state_plugins.SimSystemPosix attribute*), 1537
EIO (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EIO (*angr.state_plugins.SimSystemPosix attribute*), 1536
EISDIR (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EISDIR (*angr.state_plugins.SimSystemPosix attribute*), 1537
element_type (*angr.procedures.java_jni.array_operations.NewArray attribute*), 1338
element_type (*angr.procedures.java_jni.array_operations.NewBooleanArray attribute*), 1338
element_type (*angr.procedures.java_jni.array_operations.NewByteArray attribute*), 1338
element_type (*angr.procedures.java_jni.array_operations.NewCharArray attribute*), 1338
element_type (*angr.procedures.java_jni.array_operations.NewIntArray attribute*), 1339
element_type (*angr.procedures.java_jni.array_operations.NewLongArray attribute*), 1339
element_type (*angr.procedures.java_jni.array_operations.NewShortArray attribute*), 1339
ELFHeader (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
ELFHeader (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
ELFHeader (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
eliminatable (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegO attribute*), 335
eliminatable (*angr.analyses.data_dep.data_dependency_analysis.NodalA attribute*), 358
eliminatable (*angr.analyses.decompiler.condition_processor.AILExprId attribute*), 408
eliminatable (*angr.analyses.variable_recovery.annotations.StackLocatio attribute*), 923
eliminatable (*angr.analyses.variable_recovery.annotations.VariableSour

*attribute*), 923
eliminatable (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase attribute*), 927
eliminatable (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 1249
eliminatable (*angr.knowledge_plugins.propagations.states.RegisterAnnotation attribute*), 1274
eliminatable (*angr.knowledge_plugins.propagations.states.RegisterOutputAnnotation attribute*), 1275
eliminatable (*angr.storage.memory_mixins.address_concretization_mixin.MultiwriteAnnotation attribute*), 1633
EmFail (*angr.protos.primitives_pb2.Edge attribute*), 1403
EMFILE (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EMFILE (*angr.state_plugins.SimSystemPosix attribute*), 1537
EMLINK (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
EMLINK (*angr.state_plugins.SimSystemPosix attribute*), 1537
Empty (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 798
empty_block() (*angr.engines.pcode.lifter.IRSB static method*), 1039
EmptyBlockNotice, 710
EmptyEpsilonNFAError, 898
EmptyNodeRemover (*class in angr.analyses.decompiler.empty_node_remover*), 450
Emulator (*class in angr.emulator*), 995
EmulatorException, 996
EmulatorStopReason (*class in angr.emulator*), 996
EmWarn (*angr.protos.primitives_pb2.Edge attribute*), 1403
enable_profiling() (*in module angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 338
enable_root_logger() (*angr.misc.loggers.Loggers method*), 1316
enable_timing() (*in module angr.state_plugins.solver*), 1587
enabled_for() (*angr.analyses.decompiler.known_patterns.templates.KnownPatternTemplate method*), 532
EncodePointer (*class in angr.procedures.win32.EncodePointer*), 1382
end (*angr.keyed_region.RegionObject property*), 1116
end (*angr.rustylib.Segment attribute*), 1439
end_info_hook() (*in module angr.state_plugins.trace_additions*), 1597
Endpoint (*class in angr.protos.primitives_pb2*), 1404
endpoints (*angr.knowledge_plugins.functions.function.Function property*), 1213
endpoints_with_type (*angr.knowledge_plugins.functions.function.Function property*), 1213
ENFILE (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENFILE (*angr.state_plugins.SimSystemPosix attribute*), 1537
ENGINE_PROCESS (*angr.state_plugins.inspect.EventType attribute*), 1565
EngineException, 996
ENODEV (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENODEV (*angr.state_plugins.SimSystemPosix attribute*), 1537
ENOENT (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENOENT (*angr.state_plugins.SimSystemPosix attribute*), 1536
ENOEXEC (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENOEXEC (*angr.state_plugins.SimSystemPosix attribute*), 1536
ENOMEM (*angr.state_plugins.cgc.SimStateCGC attribute*), 1544
ENOMEM (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENOMEM (*angr.state_plugins.SimStateCGC attribute*), 1525
ENOMEM (*angr.state_plugins.SimSystemPosix attribute*), 1536
ENOSPC (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
ENOSPC (*angr.state_plugins.SimSystemPosix attribute*), 1537
ENOSYS (*angr.state_plugins.cgc.SimStateCGC attribute*), 1544
ENOSYS (*angr.state_plugins.SimStateCGC attribute*), 1525
ENOTBLK (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENOTBLK (*angr.state_plugins.SimSystemPosix attribute*), 1537
ENOTDIR (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENOTDIR (*angr.state_plugins.SimSystemPosix attribute*), 1537
ENOTTY (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENOTTY (*angr.state_plugins.SimSystemPosix attribute*), 1537
entry_state() (*angr.factory.AngrObjectFactory method*), 1110
entrypoints (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 349
Enum (*class in angr.analyses.typehoon.typeconsts*), 911

EnumVariant (*class in angr.analyses.typehoon.typeconsts*), 911
EnumVariant (*class in angr.rust.sim_type*), 1434
EnvironAtom (*class in angr.analyses.reaching_definitions.function_handler_library.stdlib*), 850
environment (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionState property*), 862
Environment (*class in angr.knowledge_plugins.key_definitions.environment*), 1247
ENXIO (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ENXIO (*angr.state_plugins.SimSystemPosix attribute*), 1536
eof() (*angr.storage.file.SimFileDescriptorBase method*), 1615
EPERM (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EPERM (*angr.state_plugins.SimSystemPosix attribute*), 1536
EPIPE (*angr.state_plugins.cgc.SimStateCGC attribute*), 1544
EPIPE (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
EPIPE (*angr.state_plugins.SimStateCGC attribute*), 1525
EPIPE (*angr.state_plugins.SimSystemPosix attribute*), 1537
EQ (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.CmpOp attribute*), 639
Eq (*class in angr.analyses.stack_pointer_tracker*), 894
Equivalence (*class in angr.analyses.typehoon.typevars*), 914
Equivalence (*class in angr.knowledge_plugins.propagations.states*), 1276
ERANGE (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
ERANGE (*angr.state_plugins.SimSystemPosix attribute*), 1537
erase() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1640
EROFS (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
EROFS (*angr.state_plugins.SimSystemPosix attribute*), 1537
err (*class in angr.procedures.libc.err*), 1353
errno (*angr.state_plugins.libc.SimStateLibc property*), 1576
ERRNO_LOCATION_FUNCS (*angr.analyses.decompiler.CStructuredCodeGenerator attribute*), 370
ERRNO_LOCATION_FUNCS (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator attribute*), 686
ERRNO_LOCATION_FUNCS (*angr.analyses.decompiler.structured_codegen.CStructuredCodeGenerator attribute*), 664
ERROR (*angr.rustylib.telemetry.StatusCode attribute*), 1321
error (*class in angr.procedures.libc.error*), 1353
error_condition() (*in module angr.state_plugins.solver*), 1587
ERROR_REG (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 981
ERROR_REG (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 980
ERROR_REG (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 982
ERROR_REG (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 982
ERROR_REG (*angr.calling_conventions.SimCCSyscall attribute*), 973
errored (*angr.sim_manager.SimulationManager property*), 1442
ErrorPropagationSimplifier (*class in angr.rust.optimization_passes.error_propagation_simplifier*), 1423
ErrorPropagationWalker (*class in angr.rust.optimization_passes.error_propagation_simplifier*), 1423
ErrorRecord (*class in angr.sim_manager*), 1447
errors (*angr.analyses.Analysis attribute*), 234
errors (*angr.knowledge_plugins.analysis.Analysis attribute*), 277
errors (*angr.Analysis attribute*), 156
ESPIPE (*angr.state_plugins.posix.SimSystemPosix attribute*), 1581
ESPIPE (*angr.state_plugins.SimSystemPosix attribute*), 1537
ESRCH (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ESRCH (*angr.state_plugins.SimSystemPosix attribute*), 1536
ETXTBSY (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
ETXTBSY (*angr.state_plugins.SimSystemPosix attribute*), 1537
eval() (*angr.state_plugins.solver.SimSolver method*), 1592
eval_atleast() (*angr.state_plugins.solver.SimSolver method*), 1594
eval_atmost() (*angr.state_plugins.solver.SimSolver method*), 1593
eval_exact() (*angr.state_plugins.solver.SimSolver method*), 1594
eval_one() (*angr.state_plugins.solver.SimSolver method*), 1592
eval_to_ast() (*angr.state_plugins.solver.SimSolver method*), 1590

eval_upto() (*angr.state_plugins.solver.SimSolver method*), 1591
evaluate_binary() (*angr.engines.pcode.behavior.OpBehavior method*), 1015
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolAnd method*), 1028
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolOr method*), 1028
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorBoolXor method*), 1027
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorEqual method*), 1016
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntAdd method*), 1019
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntAnd method*), 1023
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntCarry method*), 1020
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntDiv method*), 1025
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLeft method*), 1024
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLess method*), 1018
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntLessEqual method*), 1018
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntMult method*), 1025
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntOr method*), 1023
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntRem method*), 1026
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntRight method*), 1024
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSborrow method*), 1021
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntScarry method*), 1021
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSdiv method*), 1026
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSless method*), 1017
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSlessEqual method*), 1017
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSrem method*), 1027
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSright method*), 1024
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntSub method*), 1020
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorIntXor method*), 1022
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorNotEqual method*), 1016
evaluate_binary() (*angr.engines.pcode.behavior.OpBehaviorSubpiece method*), 1033
evaluate_unary() (*angr.engines.pcode.behavior.OpBehavior method*), 1015
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorBoolNegate method*), 1027
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorCopy method*), 1016
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorInt2Comp method*), 1022
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntNegate method*), 1022
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntSext method*), 1019
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorIntZext method*), 1019
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorLzcount method*), 1034
evaluate_unary() (*angr.engines.pcode.behavior.OpBehaviorPopcount method*), 1033
EvaluateConstConversions (*class in angr.analyses.decompiler.peephole_optimizations.evaluate_const_* 611
even_offsets_are_zero() (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy_s static method*), 582
eventsMult (*angr.state_plugins.history.SimStateHistory property*), 1562
events_of_type() (*angr.state_plugins.log.SimStateLog method*), 1577
EventType (*class in angr.state_plugins.inspect*), 1565
evict_all_cached() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingC method*), 1187
evict_all_cached() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingC method*), 1181
evict_all_cached() (*angr.knowledge_plugins.cfg.spilling_digraph.Spilli method*), 1188
evict_all_cached() (*angr.knowledge_plugins.functions.function_manag method*), 1221
evict_all_cached_edges() (*angr.knowledge_plugins.cfg.spilling_digraph.SpillingDiGraph method*), 1188
evidence (*angr.analyses.decompiler.known_patterns.GateContext attribute*), 456
evidence (*angr.analyses.decompiler.known_patterns.gating.GateContext attribute*), 511
evidence (*angr.analyses.language_detector.LanguageDetector property*), 787
exact_div_magic() (*in module angr.analyses.decompiler.known_patterns.std_vector_size*), 528
EXACTDIV_ELEMENT_SIZES (*in module angr.procedures.definitions.types_stl*), 1334
ExAllocatePool (*class in*

*angr.procedures.win32_kernel.ExAllocatePool*), 1387
EXC_COUNTER (*angr.analyses.decompiler.condition_processor.ConditionProcessor attribute*), 409
Exception (*angr.protos.primitives_pb2.Edge attribute*), 1403
exclusive() (*angr.mcp.session.ProjectSession method*), 1307
EXDEV (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
EXDEV (*angr.state_plugins.SimSystemPosix attribute*), 1537
EXECUTE (*angr.analyses.smc.TraceActions attribute*), 891
execute() (*angr.project.Project method*), 1392
execute() (*angr.sim_procedure.SimProcedure method*), 1450
ExecuteAddressGoal (*class in angr.exploration_techniques.director*), 1099
executed_instruction_count (*angr.errors.SimError attribute*), 1083
executed_instruction_count (*angr.SimError attribute*), 177
ExecveAtom (*class in angr.analyses.reaching_definitions.function_handler_library.stdlib*), 850
ExFreePoolWithTag (*class in angr.procedures.win32_kernel.ExFreePoolWithTag*), 1387
Existence (*class in angr.analyses.typehoon.typevars*), 915
exists() (*angr.knowledge_plugins.propagations.propagation_manager.PropagationManager method*), 1270
exists_in_replacements() (*in module angr.analyses.cfg.indirect_jump_resolvers.const_resolver*), 332
EXIT (*angr.emulator.EmulatorStopReason attribute*), 996
EXIT (*angr.EmulatorStopReason attribute*), 161
EXIT (*angr.state_plugins.inspect.EventType attribute*), 1565
Exit (*class in angr.procedures.java_lang.exit*), 1348
exit (*class in angr.procedures.libc.exit*), 1353
exit() (*angr.sim_procedure.SimProcedure method*), 1452
exit_guard (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
exit_guard (*angr.state_plugins.InspectAttrs attribute*), 1498
exit_hook() (*in module angr.state_plugins.trace_additions*), 1597
exit_jumpkind (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
exit_jumpkind (*angr.state_plugins.InspectAttrs attribute*), 1498
exit_observe() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 869
exit_statements (*angr.engines.pcode.lifter.IRSB property*), 1039
exit_target (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
exit_target (*angr.state_plugins.InspectAttrs attribute*), 1498
ExitProcess (*class in angr.procedures.win32.ExitProcess*), 1382
ExitStatement (*class in angr.engines.pcode.lifter*), 1037
ExplicitFillerMixin (*class in angr.storage.memory_mixins.default_filler_mixin*), 1635
ExplorationStatusNotifier (*class in angr.distributed.worker*), 994
ExplorationTechnique (*class in angr.exploration_techniques.base*), 1097
explore() (*angr.sim_manager.SimulationManager method*), 1443
Explorer (*class in angr.exploration_techniques.explorer*), 1102
export_serialized() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionManager method*), 1220
export_serialized() (*angr.knowledge_plugins.structured_code.SpillingDecompilationCache method*), 1279
export_serialized_nodes() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1183
EXPR (*angr.state_plugins.inspect.EventType attribute*), 1565
expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
expr (*angr.state_plugins.InspectAttrs attribute*), 1498
expr_anchor_key() (*in module angr.analyses.decompiler.known_patterns.dsl*), 506
expr_classes (*angr.analyses.decompiler.peephole_optimizations.a_div_c attribute*), 600
expr_classes (*angr.analyses.decompiler.peephole_optimizations.a_mul_c attribute*), 600
expr_classes (*angr.analyses.decompiler.peephole_optimizations.a_mul_c attribute*), 600
expr_classes (*angr.analyses.decompiler.peephole_optimizations.a_shl_c attribute*), 601
expr_classes (*angr.analyses.decompiler.peephole_optimizations.a_sub_a attribute*), 601
expr_classes (*angr.analyses.decompiler.peephole_optimizations.a_sub_a attribute*), 601
expr_classes (*angr.analyses.decompiler.peephole_optimizations.a_sub_a attribute*), 601
expr_classes (*angr.analyses.decompiler.peephole_optimizations.arm_cm

*attribute*), 602
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveNegation attribute*), 604
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveNegationEq attribute*), 605
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveNegationOr attribute*), 605
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveOrConstant attribute*), 605
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveOrEqualities attribute*), 606
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveShiftNarrowing attribute*), 606
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveShiftOnMaskedShift attribute*), 606
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveShiftOnMaskedShift attribute*), 608
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveSubConstants attribute*), 608
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveSubZero attribute*), 609
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveTrivialConversions attribute*), 610
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveTrivialConversions attribute*), 610
expr_classes (*angr.analyses.decompiler.peephole_optimizations.RemoveZeroExtensionOnSmallVariable attribute*), 610
expr_classes (*angr.analyses.decompiler.peephole_optimizations.ShlSubShl attribute*), 611
expr_classes (*angr.analyses.decompiler.peephole_optimizations.SingleBitTest attribute*), 611
expr_classes (*angr.analyses.decompiler.peephole_optimizations.SingleBitTest attribute*), 611
expr_classes (*angr.analyses.decompiler.peephole_optimizations.TidyStackAddr attribute*), 612
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 612
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 612
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 613
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 613
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 613
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 614
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 614
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 615
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 615
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 615
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 616
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 616
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 616
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 617
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 617
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 617
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 617
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 618
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 618
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 618
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 619
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 620
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 620
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 620
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 620
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 620
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 621
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 621
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 735
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 736
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 731
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 732
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 739
expr_classes (*angr.analyses.decompiler.peephole_optimizations.UnifyReferenceExpressions attribute*), 733
expr_handlers (*angr.misc.ailment.AILBlockWalker property*), 215
expr_result_needed() (*angr.state_plugins.inspect.InspectAttributes attribute*), 1567
expr_result_used() (*angr.state_plugins.symbolic_memory.BitmaskedSymbolicMemory*

1498
Expression (*class in angr.ailment.expression*), 220
ExpressionCounter (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 636
ExpressionFolder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 637
ExpressionLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 634
ExpressionNarrower (*class in angr.analyses.decompiler.expression_narrower*), 451
ExpressionReplacer (*class in angr.analyses.decompiler.optimization_passes.expr_op_swapper*), 575
ExpressionReplacer (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter*), 582
ExpressionReplacer (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 636
expressions (*angr.engines.pcode.lifter.IRSB property*), 1040
ExpressionSequenceWalker (*class in angr.analyses.decompiler.optimization_passes.peephole_simplifier*), 589
ExpressionSpotter (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 636
ExpressionUseFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 635
ExprNarrowingInfo (*class in angr.analyses.decompiler.expression_narrower*), 451
ExprOpSwapper (*class in angr.analyses.decompiler.optimization_passes.expr_op_swapper*), 576
extend() (*angr.engines.pcode.lifter.IRSB method*), 1040
extend() (*angr.utils.ins_addr_list.InsAddrList method*), 1690
extend_actions() (*angr.state_plugins.history.SimStateHistory method*), 1562
extend_actions() (*angr.state_plugins.log.SimStateLog method*), 1577
extend_ail_expression() (*angr.knowledge_plugins.propagations.prop_value.PropValue static method*), 1270
extend_size() (*angr.engines.vex.claripy.irop.SimIROp method*), 1074
ExtendedByteAndMask (*class in angr.analyses.decompiler.peephole_optimizations.extended_byte_and_mask*), 612
extern (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatch attribute*), 1246
extern_idx (*angr.code_location.AILCodeLocation property*), 986
extern_idx_unwrap (*angr.code_location.AILCodeLocation property*), 986
external_entry_count() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 627
external_labels (*angr.analyses.decompiler.known_patterns.dsl.PGraph property*), 506
ExternalCodeLocation (*class in angr.code_location*), 985
ExternalFunction (*class in angr.protos.primitives_pb2*), 1404
ExternalVariable (*class in angr.protos.primitives_pb2*), 1404
ExtraBytes (*class in angr.protos.clinic_pb2*), 1397
EXTRA_ARCHES (*angr.calling_conventions.SimCC attribute*), 967
EXTRA_ARCHES (*angr.calling_conventions.SimCCARMHF attribute*), 977
EXTRA_ARCHES (*angr.SimCC attribute*), 173
extra_args (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 457
extra_args (*angr.analyses.decompiler.known_patterns.pattern.KnownPat attribute*), 521
extra_args (*angr.analyses.decompiler.KnownPattern attribute*), 379
Extract (*class in angr.ailment.expression*), 220
extract() (*angr.procedures.win_user32.messagebox.MessageBoxA method*), 1388
extract() (*angr.procedures.win_user32.messagebox.MessageBoxW method*), 1388
extract() (*angr.sim_type.SimCppClass method*), 1475
extract() (*angr.sim_type.SimStruct method*), 1471
extract() (*angr.sim_type.SimType method*), 1460
extract() (*angr.sim_type.SimTypeArray method*), 1468
extract() (*angr.sim_type.SimTypeBitfield method*), 1475
extract() (*angr.sim_type.SimTypeBool method*), 1465
extract() (*angr.sim_type.SimTypeChar method*), 1464
extract() (*angr.sim_type.SimTypeEnum method*), 1473
extract() (*angr.sim_type.SimTypeFd method*), 1466
extract() (*angr.sim_type.SimTypeFloat method*), 1470
extract() (*angr.sim_type.SimTypeInt method*), 1463
extract() (*angr.sim_type.SimTypeNum method*), 1463
extract() (*angr.sim_type.SimTypeNumOffset method*), 1475
extract() (*angr.sim_type.SimTypePointer method*), 1467
extract() (*angr.sim_type.SimTypeReference method*), 1467

extract() (*angr.sim_type.SimTypeString method*), 1468
extract() (*angr.sim_type.SimTypeWideChar method*), 1465
extract() (*angr.sim_type.SimTypeWString method*), 1469
extract() (*angr.sim_type.SimUnion method*), 1472
extract() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 1650
extract_ail_expression() (*angr.knowledge_plugins.propagations.prop_value.PropValue static method*), 1270
extract_callee() (*in module angr.rust.optimization_passes.utils*), 1430
extract_claripy() (*angr.sim_type.SimType method*), 1460
extract_components() (*angr.procedures.stubs.format_parser.FormatParser method*), 1379
extract_defs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 862
extract_defs() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1251
extract_defs_from_annotations() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1252
extract_defs_from_mv() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1252
extract_jump_targets() (*in module angr.analyses.decompiler.utils*), 720
extract_offset_to_sp() (*angr.engines.light.engine.SimEngineLight static method*), 1011
extract_partial_expr() (*in module angr.utils.ail*), 1670
extract_scrutinee_and_discriminant() (*angr.rust.optimization_passes.pre_pattern_match_simplifier.PrePatternMatchSimplifier static method*), 1427
extract_stack_data_flow() (*angr.rust.mixins.dfa_mixin.DFAMixin static method*), 1419
extract_stack_to_reg_data_flow() (*angr.rust.mixins.dfa_mixin.DFAMixin static method*), 1419
extract_str() (*in module angr.rust.optimization_passes.utils*), 1430
extract_str_from_addr() (*in module angr.rust.optimization_passes.utils*), 1430
extract_terms() (*in module angr.analyses.decompiler.structured_codegen.c*), 670
extract_terms() (*in module angr.analyses.decompiler.structured_codegen.rust*), 691
extract_value_if_concrete() (*angr.analyses.xrefs.SimEngineXRefsVEX static method*), 937
extract_variables() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase static method*), 928
extract_vars_and_offset() (*in module angr.rust.utils.ail*), 1438
extract_write_to_stack_vvar() (*angr.rust.mixins.dfa_mixin.DFAMixin static method*), 1419

## F

FactCollector (*class in angr.analyses.calling_convention.fact_collector*), 288
FactCollector (*class in angr.rust.analyses.rust_calling_convention.fact_collector*), 1405
FactCollectorState (*class in angr.analyses.calling_convention.fact_collector*), 286
FactData (*in module angr.analyses.calling_convention.fact_collector*), 286
FAILURE (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Case2RegResolver attribute*), 338
FAILURE (*angr.emulator.EmulatorStopReason attribute*), 996
FAILURE (*angr.EmulatorStopReason attribute*), 161
FakeReturn (*angr.protos.primitives_pb2.Edge attribute*), 1403
FAST (*angr.analyses.complete_calling_conventions.CallingConventionAnalysis attribute*), 353
fast_memory_load() (*angr.analyses.reassembler.Reassembler method*), 878
FastConstantPropagation (*class in angr.analyses.fcp.fcp*), 746
FasterMemory (*class in angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier*), 579
FASTISH (*angr.analyses.complete_calling_conventions.CallingConventionAnalysis attribute*), 353
FastMemory (*class in angr.storage.memory_mixins*), 1619
fclose (*class in angr.procedures.libc.fclose*), 1354
fcntl (*class in angr.procedures.posix.fcntl*), 1371
FCPState (*class in angr.analyses.fcp.fcp*), 744
Fd (*class in angr.analyses.typehoon.typeconsts*), 908
FD_SETSIZE (*angr.state_plugins.cgc.SimStateCGC attribute*), 1544
FD_SETSIZE (*angr.state_plugins.SimStateCGC attribute*), 1525
FDAtom (*class in angr.analyses.reaching_definitions.function_handler_library*), 855

fdopen (*class in angr.procedures.posix.fdopen*), 1371
fdprintf (*class in angr.analyses.identifier.functions.fdprintf*), 772
fdwait (*class in angr.procedures.cgc.fdwait*), 1323
FEATURE_2BYTE_CTYPE (*angr.analyses.flirt.consts.FlirtFeatureFlag attribute*), 754
FEATURE_ALT_CTYPE_CRC (*angr.analyses.flirt.consts.FlirtFeatureFlag attribute*), 755
FEATURE_COMPRESSED (*angr.analyses.flirt.consts.FlirtFeatureFlag attribute*), 755
FEATURE_CTYPE_CRC (*angr.analyses.flirt.consts.FlirtFeatureFlag attribute*), 754
FEATURE_STARTUP (*angr.analyses.flirt.consts.FlirtFeatureFlag attribute*), 754
feof (*class in angr.procedures.libc.feof*), 1354
feof_unlocked (*in module angr.procedures.libc.feof*), 1354
FetchingZeroPageError, 1603
fflush (*class in angr.procedures.libc.fflush*), 1354
fflush_unlocked (*in module angr.procedures.libc.fflush*), 1354
ffs() (*in module angr.utils.bits*), 1674
fgetc (*class in angr.procedures.libc.fgetc*), 1354
fgetc_unlocked (*in module angr.procedures.libc.fgetc*), 1354
fgets (*class in angr.procedures.libc.fgets*), 1354
fgets_unlocked (*in module angr.procedures.libc.fgets*), 1354
field_offsets (*angr.rust.sim_type.EnumVariant property*), 1434
FieldReferenceCleanup (*class in angr.analyses.decompiler.structured_codegen.c*), 687
FieldReferenceCleanup (*class in angr.analyses.decompiler.structured_codegen.rust*), 709
fields (*angr.sim_type.SimUnion attribute*), 1472
FILE_AIXAR (*angr.analyses.flirt.consts.FlirtFileType attribute*), 754
FILE_AOUT (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_AR (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_BIN (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_COFF (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_DOS_COM (*angr.analyses.flirt.consts.FlirtFileType attribute*), 754
FILE_DOS_COM_OLD (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_DOS_EXE (*angr.analyses.flirt.consts.FlirtFileType attribute*), 754
FILE_DOS_EXE_OLD (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_DOSDRV (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_ELF (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
file_exists (*angr.storage.file.SimFileDescriptor property*), 1615
file_exists (*angr.storage.file.SimFileDescriptorBase property*), 1615
FILE_INTELHEX (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_LE (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_LOADER (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_LX (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_MOSHEX (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_NE (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_NLM (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_OMF (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_OMFLIB (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_PE (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_PILOT (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_SREC (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_W32RUN (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
FILE_ZIP (*angr.analyses.flirt.consts.FlirtFileType attribute*), 753
fileno (*class in angr.procedures.posix.fileno*), 1371
fileno_unlocked (*in module angr.procedures.posix.fileno*), 1371
fill_concrete() (*angr.procedures.win32.GetLastInputInfo.GetLastInput method*), 1382
fill_concrete() (*angr.procedures.win32.GetProcessAffinityMask.GetPr method*), 1383
fill_content() (*angr.knowledge_plugins.cfg.memory_data.MemoryDat method*), 1179
fill_from_timestamp() (*angr.procedures.win32.sim_time.GetLocalTime method*), 1386
fill_from_timestamp() (*angr.procedures.win32.sim_time.GetSystemTimeAsFileTime method*), 1386

fill_reg_map() (*in module angr.analyses.reassembler*), 870
fill_symbolic() (*angr.procedures.win32.GetLastInputInfo.GetLastInputInfo method*), 1382
fill_symbolic() (*angr.procedures.win32.GetProcessAffinityMask method*), 1383
fill_symbolic() (*angr.procedures.win32.sim_time.GetLocalTime method*), 1386
fill_symbolic() (*angr.procedures.win32.sim_time.GetSystemTimeAsFileTime method*), 1386
filter() (*angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJumpResolver method*), 340
filter() (*angr.analyses.veritesting.CallTracingFilter method*), 931
filter() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1097
filter() (*angr.sim_manager.SimulationManager method*), 1444
filter_actions() (*angr.state_plugins.history.SimStateHistory method*), 1561
filter_cond_regions() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 640
filter_constraints() (*angr.state_plugins.trace_additions.ZenPlugin method*), 1598
filter_module() (*in module angr.misc.autoimport*), 1314
filter_replacements() (*angr.knowledge_plugins.propagations.states.PropagatorState method*), 1274
filter_scopes() (*angr.analyses.deobfuscator.scope_ops_analyzer.ScopeOpsAnalyzer method*), 738
filtered() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 632
finalize() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 630
finalize() (*angr.analyses.decompiler.ssailification.traversal_engine_vex.SimEngineSSATraversal method*), 657
find() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1638
find() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 1663
find() (*angr.storage.memory_mixins.regioned_memory.static_find_mixin.StaticFindMixin method*), 1664
find() (*angr.storage.memory_mixins.smart_find_mixin.SmartFindMixin method*), 1666
find() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 1666
find_all_predecessors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 837
find_all_successors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 837
find_backward_path() (*angr.rustylib.analyses.rust_calling_convention.pathfinder.Pathfinder method*), 1409
find_basic_blocks_between() (*angr.analyses.pathfinder.Pathfinder method*), 795
find_block_by_addr() (*in module angr.analyses.decompiler.utils*), 726
find_block_by_addr_and_idx() (*in module angr.analyses.decompiler.utils*), 726
find_block_by_similarity() (*in module angr.analyses.decompiler.optimization_passes.duplication_reverter*), 572
find_block_in_successors_by_addr() (*in module angr.analyses.decompiler.optimization_passes.duplication_reverter*), 572
find_call() (*in module angr.rust.utils.ail*), 1437
find_cc() (*angr.calling_conventions.SimCC static method*), 970
find_conflicting_call_args() (*in module angr.analyses.decompiler.optimization_passes.const_prop_reverter static method*), 564
find_consumers() (*angr.analyses.ddg.DDG method*), 366
find_data_references_and_update_memory_data() (*angr.analyses.decompiler.decompiler.Decompiler method*), 415
find_declaration() (*angr.knowledge_plugins.functions.function.Function method*), 1217
find_definition() (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase method*), 604
find_definitions() (*angr.analyses.ddg.DDG method*), 366
find_definitions() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 836
find_defs_at() (*angr.knowledge_plugins.key_definitions.liveness.LivenessAnalysis method*), 1258
find_defs_at() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 1259
find_defs_at_raw() (*angr.knowledge_plugins.key_definitions.liveness.LivenessAnalysis method*), 1258
find_forward_path() (*angr.rustylib.analyses.rust_calling_convention.pathfinder.Pathfinder method*), 1409
find_function_for_reflow_into_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1173
find_functions_by_pattern() (*in module angr.mcp.server*), 1306
find_go_noreturn_functions() (*in module angr.utils.go_runtime*), 1682
find_go_stack_check_preambles() (*in module angr.analyses.cfg.go_prologue*), 327

find_goto_edges() (*angr.analyses.decompiler.goto_manager.GotoManager method*), 452
find_killers() (*angr.analyses.ddg.DDG method*), 366
find_merge_points() (*angr.utils.graph.GraphUtils static method*), 1685
find_path() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 837
find_paths() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 837
find_reg_ptr_to_reg_data_flow() (*angr.rust.mixins.dfa_mixin.DFAMixin method*), 1419
find_ret2arg0_paths() (*angr.rust.analyses.rust_calling_convention.pathfinder.PathFinder method*), 1409
find_sources() (*angr.analyses.ddg.DDG method*), 366
find_stack_data_flow() (*angr.rust.mixins.dfa_mixin.DFAMixin method*), 1419
find_stack_to_reg_data_flow() (*angr.rust.mixins.dfa_mixin.DFAMixin method*), 1419
find_stack_vars_x86() (*angr.analyses.identifier.identify.Identifier method*), 785
find_successor() (*in module angr.analyses.pathfinder*), 795
find_type_refs() (*in module angr.utils.types*), 1702
find_variable_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1285
find_variable_by_stmt() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1284
find_variables_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1285
find_variables_by_insn() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1284
find_variables_by_register() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1285
find_variables_by_stack_offset() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1285
find_variables_by_stmt() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1284
find_widening_points() (*angr.utils.graph.GraphUtils static method*), 1686
FindCallsTo (*class in angr.analyses.deobfuscator.hash_lookup_api_deobfuscator*), 727
FindClass (*class in angr.procedures.java_jni.class_and_interface_operations*), 1340
FindExtraDefs (*class in angr.utils.ssa.vvar_extra_defs_collector*), 1409
FindFirstNodeInSet (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 641
finish() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
fire() (*angr.state_plugins.inspect.BP method*), 1571
first_field_offset (*angr.rust.sim_type.EnumVariant property*), 1434
first_label() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 918
first_line (*angr.knowledge_plugins.comments.Comment property*), 1190
first_non_label_stmt() (*angr.rust.mixins.cfa_mixin.CFAMixin static method*), 1417
first_nonlabel_node() (*in module angr.analyses.decompiler.utils*), 723
first_nonlabel_nonphi_node() (*in module angr.analyses.decompiler.utils*), 723
first_nonlabel_nonphi_statement() (*in module angr.analyses.decompiler.utils*), 722
first_nonlabel_statement() (*in module angr.analyses.decompiler.utils*), 722
first_nonlabel_statement_id() (*in module angr.analyses.decompiler.utils*), 722
fix_prototype_returnty() (*angr.sim_procedure.SimProcedure method*), 1451
fix_stack_vvar_uses() (*angr.rust.mixins.ssa_variable_mixin.SSAVariableMixin method*), 1420
FixedBuffer (*class in angr.analyses.decompiler.optimization_passes.static_vvar_rewriter*), 594
FixedBufferMsg (*class in angr.protos.ail_types_pb2*), 1394
FixedBufferPtr (*class in angr.analyses.decompiler.optimization_passes.static_vvar_rewriter*), 595
FixedBufferPtrMsg (*class in angr.protos.ail_types_pb2*), 1394
fixup_offsets() (*angr.sim_type.SimStruct method*), 1471
fixup_test() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 778
fixup_test() (*angr.analyses.identifier.functions.recv_until.receive_until_buffer method*), 777
flags (*angr.procedures.stubs.format_parser.FormatParser*

*attribute*), 1379
Flags (*class in angr.storage.file*), 1609
flatten_typevar() (*angr.analyses.typehoon.simple_solver.Sketch static method*), 901
FlipBooleanCmp (*class in angr.analyses.decompiler.optimization_passes.flip_boolean_cmp*), 576
FlipBooleanWalker (*class in angr.analyses.decompiler.optimization_passes.flip_boolean_cmp*), 576
flirt_arch_to_arch_name() (*in module angr.analyses.flirt*), 750
flirt_os_type_to_os_name() (*in module angr.analyses.flirt*), 750
FlirtAnalysis (*class in angr.analyses.flirt.flirt*), 755
FlirtAppType (*class in angr.analyses.flirt.consts*), 754
FlirtArch (*class in angr.analyses.flirt.consts*), 751
FlirtFeatureFlag (*class in angr.analyses.flirt.consts*), 754
FlirtFileType (*class in angr.analyses.flirt.consts*), 753
FlirtFunction (*class in angr.analyses.flirt.flirt_function*), 756
FlirtFunctionFlag (*class in angr.analyses.flirt.consts*), 755
FlirtMatcher (*class in angr.analyses.flirt.flirt_matcher*), 756
FlirtModule (*class in angr.analyses.flirt.flirt_module*), 757
FlirtNode (*class in angr.analyses.flirt.flirt_node*), 757
FlirtOSType (*class in angr.analyses.flirt.consts*), 754
FlirtParseFlag (*class in angr.analyses.flirt.consts*), 755
FlirtSignature (*class in angr.analyses.flirt.flirt_sig*), 758
FlirtSignatureError, 758
FlirtSignatureParsed (*class in angr.analyses.flirt.flirt_sig*), 758
FlirtSigPropagation (*class in angr.rust.analyses.flirt_sig_propagation*), 1407
FLOAT (*angr.analyses.decompiler.structured_codegen.base.CConstantType attribute*), 668
Float (*class in angr.analyses.typehoon.typeconsts*), 909
Float32 (*class in angr.analyses.typehoon.typeconsts*), 909
Float64 (*class in angr.analyses.typehoon.typeconsts*), 909
float_len_mod (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1379
float_len_mod (*angr.procedures.stubs.format_parser.ScanfFormatParser attribute*), 1380
float_spec (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1379
float_type() (*in module angr.analyses.typehoon.typeconsts*), 913
FloatingPoint (*angr.knowledge_plugins.cfg.memory_data.MemoryDataS attribute*), 1178
FloatingPoint (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
FloatingPoint (*angr.protos.cfg_pb2.MemoryData attribute*), 1395
floor_addr() (*angr.analyses.cfg.cfb.CFBlanket method*), 305
floor_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1168
floor_addr() (*angr.knowledge_plugins.functions.function_manager.Funct method*), 1219
floor_addr() (*angr.knowledge_plugins.functions.function_manager.Funct method*), 1218
floor_addr() (*angr.knowledge_plugins.functions.function_manager.Funct method*), 1223
floor_func() (*angr.knowledge_plugins.functions.function_manager.Funct method*), 1223
floor_item() (*angr.analyses.cfg.cfb.CFBlanket method*), 305
floor_items() (*angr.analyses.cfg.cfb.CFBlanket method*), 305
FlsAlloc (*class in angr.procedures.win32.local_storage*), 1385
FlsFree (*class in angr.procedures.win32.local_storage*), 1385
FlsGetValue (*class in angr.procedures.win32.local_storage*), 1385
FlsSetValue (*class in angr.procedures.win32.local_storage*), 1385
flush_pages() (*angr.storage.memory_mixins.paged_memory.paged_mem method*), 1642
FlushDCache (*angr.protos.primitives_pb2.Edge attribute*), 1403
fmt (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 681
fmt (*angr.analyses.decompiler.structured_codegen.rust.RustConstant property*), 704
fmt_char (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 681
fmt_char (*angr.analyses.decompiler.structured_codegen.rust.RustConstant property*), 704
fmt_double (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 682
fmt_float (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 682
fmt_hex (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 681
fmt_hex (*angr.analyses.decompiler.structured_codegen.rust.RustConstant property*), 704
fmt_int() (*angr.analyses.decompiler.structured_codegen.c.CConstant method*), 682
fmt_int() (*angr.analyses.decompiler.structured_codegen.rust.RustConsta

*method*), 704
fmt_neg (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 681
fmt_neg (*angr.analyses.decompiler.structured_codegen.rust.RustFunctionCall property*), 704
fopen (*class in angr.procedures.libc.fopen*), 1354
force_unroll_loops() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 313
forget_last_label() (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode method*), 902
FORGOTTEN (*class in angr.analyses.typehoon.simple_solver*), 901
FORK (*angr.state_plugins.inspect.EventType attribute*), 1565
fork (*class in angr.procedures.posix.fork*), 1372
format() (*angr.analyses.analysis.AnalysisLogEntry method*), 275
format_wrappers (*angr.rust.optimization_passes.macro.format_macro_simplifier.FormatWrapperIdentification property*), 1424
FormatInfo (*class in angr.state_plugins.trace_additions*), 1596
FormatInfoDontConstrain (*class in angr.state_plugins.trace_additions*), 1597
FormatInfoIntToStr (*class in angr.state_plugins.trace_additions*), 1596
FormatInfoStrToInt (*class in angr.state_plugins.trace_additions*), 1596
FormatMacroSimplifier (*class in angr.rust.optimization_passes.macro.format_macro_simplifier*), 1424
FormatParser (*class in angr.procedures.stubs.format_parser*), 1379
FormatSpecifier (*class in angr.procedures.stubs.format_parser*), 1379
FormatString (*class in angr.procedures.stubs.format_parser*), 1378
FormatWrapperIdentification (*class in angr.rust.optimization_passes.macro.format_macro_simplifier*), 1423
FormatWrappers (*class in angr.rust.optimization_passes.macro.format_macro_simplifier*), 1424
ForwardAnalysis (*class in angr.analyses.forward_analysis.forward_analysis*), 763
ForwardAnalysisForClinic (*class in angr.analyses.forward_analysis.forward_analysis*), 764
ForwardAnalysisForDummies (*class in angr.analyses.forward_analysis.forward_analysis*), 764
FourOrFive (*class in angr.analyses.identifier.functions.int2str*), 773
FP_ARG_REGS (*angr.calling_conventions.SimCC attribute*), 967
FP_ARG_REGS (*angr.calling_conventions.SimCCAArch64 attribute*), 978
FP_ARG_REGS (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 978
FP_ARG_REGS (*angr.calling_conventions.SimCCAArch64WindowsSyscall attribute*), 979
FP_ARG_REGS (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 976
FP_ARG_REGS (*angr.calling_conventions.SimCCARM attribute*), 976
FP_ARG_REGS (*angr.calling_conventions.SimCCARMHF attribute*), 977
FP_ARG_REGS (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 977
FP_ARG_REGS (*angr.calling_conventions.SimCCARMWindowsSyscall attribute*), 978
FP_ARG_REGS (*angr.calling_conventions.SimCCCDecl attribute*), 971
FP_ARG_REGS (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
FP_ARG_REGS (*angr.calling_conventions.SimCCGoAMD64ABI0 attribute*), 976
FP_ARG_REGS (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
FP_ARG_REGS (*angr.calling_conventions.SimCCN64 attribute*), 980
FP_ARG_REGS (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 981
FP_ARG_REGS (*angr.calling_conventions.SimCCO32 attribute*), 979
FP_ARG_REGS (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 980
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC attribute*), 981
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC64 attribute*), 982
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 982
FP_ARG_REGS (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 981
FP_ARG_REGS (*angr.calling_conventions.SimCCRISCV64 attribute*), 979
FP_ARG_REGS (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 979
FP_ARG_REGS (*angr.calling_conventions.SimCCS390X attribute*), 983
FP_ARG_REGS (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 983
FP_ARG_REGS (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
FP_ARG_REGS (*angr.calling_conventions.SimCCX86LinuxSyscall*

*attribute*), 974
FP_ARG_REGS (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 974
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCM68k attribute*), 1034
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 1035
FP_ARG_REGS (*angr.engines.pcode.cc.SimCCXtensa attribute*), 1036
FP_ARG_REGS (*angr.SimCC attribute*), 173
fp_args (*angr.calling_conventions.SimCC property*), 968
FP_RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 967
FP_RETURN_VAL (*angr.calling_conventions.SimCCARMHF attribute*), 977
FP_RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 971
FP_RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
FP_RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64ABI0 attribute*), 976
FP_RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
FP_RETURN_VAL (*angr.calling_conventions.SimCCRISCV64 attribute*), 979
FP_RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
FP_RETURN_VAL (*angr.SimCC attribute*), 173
fprintf (*class in angr.procedures.libc.fprintf*), 1354
fputc (*class in angr.procedures.libc.fputc*), 1355
fputc_unlocked (*in module angr.procedures.libc.fputc*), 1355
fputs (*class in angr.procedures.libc.fputs*), 1355
fputs_unlocked (*in module angr.procedures.libc.fputs*), 1355
frame (*angr.engines.ail.engine_light.SimEngineAILSimState property*), 1003
fread (*class in angr.procedures.libc.fread*), 1355
fread_unlocked (*in module angr.procedures.libc.fread*), 1355
free (*class in angr.analyses.identifier.functions.free*), 773
free (*class in angr.procedures.libc.free*), 1355
free() (*angr.analyses.reaching_definitions.heap_allocator.HeapAllocator method*), 857
free() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 1558
free_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 1558
freeze() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 895
fresh_block() (*angr.factory.AngrObjectFactory method*), 1113
fresh_constraints (*angr.state_plugins.log.SimStateLog property*), 1577
FrozenVirtualVariableRewriter (*class in angr.analyses.decompiler.optimization_passes.return_duplicator_*592
from_addr_list() (*angr.utils.ins_addr_list.InsAddrList class method*), 1690
from_ail_expr() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 1242
from_argument() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 1242
from_bits (*angr.analyses.decompiler.known_patterns.dsl.PConv attribute*), 495
from_bits (*angr.analyses.decompiler.known_patterns.PConv attribute*), 470
from_block_lcs() (*angr.analyses.decompiler.optimization_passes.duplic class method*), 568
from_bytes() (*angr.ailment.Block class method*), 210
from_bytes() (*angr.ailment.block.Block class method*), 215
from_bytes() (*angr.ailment.expression.Expression static method*), 220
from_bytes() (*angr.ailment.statement.Statement static method*), 226
from_cle_variable() (*angr.state_plugins.debug_variables.SimDebugVariable static method*), 1545
from_codeloc() (*angr.code_location.AILCodeLocation static method*), 986
from_digraph() (*angr.annocfg.AnnotatedCFG method*), 957
from_env() (*angr.llm_client.LLMClient class method*), 1294
from_field_id() (*angr.engines.soot.values.instancefieldref.SimSootValue class method*), 1060
from_field_id() (*angr.engines.soot.values.staticfieldref.SimSootValue_S class method*), 1061
from_json() (*angr.analyses.decompiler.notes.decompilation_note.Decomp class method*), 538
from_json() (*angr.analyses.decompiler.optimization_passes.expr_op_swa class method*), 576
from_json() (*angr.analyses.decompiler.variable_map.VariableMap class method*), 731
from_json() (*angr.angrdb.serializers.structured_code.ConstFormatsSeria static method*), 954
from_json() (*angr.knowledge_plugins.functions.function.FunctionInfo class method*), 1209
from_json() (*angr.knowledge_plugins.functions.function_parser.CallingC static method*), 1227
from_json() (*angr.procedures.definitions.SimLibrary static method*), 1325
from_json() (*angr.procedures.definitions.SimTypeCollection class method*), 1325
from_json() (*angr.rust.sim_type.EnumVariant static method*), 1434

from_json() (*angr.sim_type.SimType static method*), 1461
from_jsonable() (*angr.analyses.decompiler.notes.decompilation_notes.DecompilationNote class method*), 538
from_jsonable() (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.OpDescriptor class method*), 575
from_name() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 1193
from_name_and_pc() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 1193
from_networkx() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1186
from_no_data() (*angr.rust.sim_type.EnumVariant static method*), 1434
from_pc() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 1192
from_project() (*angr.analyses.decompiler.known_patterns.context.PatternContext class method*), 488
from_signature (*angr.knowledge_plugins.functions.function.Function property*), 1211
from_single_field_ty() (*angr.rust.sim_type.EnumVariant static method*), 1434
from_sootvalue() (*angr.engines.soot.values.arrayref.SimSootValue_ArrayRef class method*), 1059
from_sootvalue() (*angr.engines.soot.values.arrayref.SimSootValue_ArrayRef class method*), 1059
from_sootvalue() (*angr.engines.soot.values.base.SimSootValue_ class method*), 1060
from_sootvalue() (*angr.engines.soot.values.constants.SimSootValue_InstanceConstant class method*), 1060
from_sootvalue() (*angr.engines.soot.values.instancefieldref.SimSootValue_InstanceFieldRef class method*), 1060
from_sootvalue() (*angr.engines.soot.values.local.SimSootValue_Local class method*), 1060
from_sootvalue() (*angr.engines.soot.values.paramref.SimSootValue_ParamRef class method*), 1061
from_sootvalue() (*angr.engines.soot.values.staticfieldref.SimSootValue_StaticFieldRef class method*), 1061
from_sootvalue() (*angr.engines.soot.values.strref.SimSootValue_StrropRef class method*), 1061
from_sootvalue() (*angr.engines.soot.values.thisref.SimSootValue_ThisRef class method*), 1062
from_state() (*angr.analyses.variable_recovery.annotations.VariableSourceAnnotation static method*), 923
from_value_and_details() (*angr.knowledge_plugins.propagations.prop_value.PropValue static method*), 1270
from_value_and_labels() (*angr.knowledge_plugins.propagations.prop_value.PropValue static method*), 1269
frontier_locs (*angr.analyses.decompiler.known_patterns.KnownPatternMatch attribute*), 507
frontier_locs (*angr.analyses.decompiler.known_patterns.KnownPatternMatch attribute*), 461
frontier_locs (*angr.analyses.decompiler.KnownPatternMatch attribute*), 383
FrozenStackPointerTrackerState (*class in angr.analyses.stack_pointer_tracker*), 894
fscanf (*class in angr.procedures.libc.fscanf*), 1355
fseek (*class in angr.procedures.libc.fseek*), 1355
fseeko (*in module angr.procedures.libc.fseek*), 1355
fstat (*class in angr.procedures.linux_kernel.fstat*), 1365
fstat() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
fstat64 (*class in angr.procedures.linux_kernel.fstat64*), 1365
fstat_with_result() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
ftell (*class in angr.procedures.libc.ftell*), 1355
ftello (*in module angr.procedures.libc.ftell*), 1355
full_cmpr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 673
full_graph (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 627
full_init_state() (*angr.factory.AngrObjectFactory method*), 1111
full_simplify() (*angr.state_hierarchy.StateHierarchy method*), 1494
full_view (*angr.analyses.decompiler.region_overlay.RegionOverlayGraph property*), 631
FSnc (*class in angr.analyses.identifier.func*), 771
func_addr (*angr.analyses.cfg.cfg_job_base.CFGJobBase property*), 326
func_addr (*angr.analyses.decompiler.edits.EditResult attribute*), 424
func_addr (*angr.analyses.decompiler.edits.results.EditResult attribute*), 450
func_addr (*angr.analyses.variable_recovery.engine_base.SimEngineVRBase property*), 924
func_addr (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecovery property*), 929
func_addr (*angr.knowledge_plugins.cfg.block_id.BlockID property*), 1165
func_edge_type_from_pb() (*in module angr.utils.enums_conv*), 1679
func_edge_type_to_pb() (*in module angr.utils.enums_conv*), 1679
func_graph (*angr.analyses.reaching_definitions.subject.Subject property*), 870
func_lookup() (*angr.analyses.disassembly.Disassembly method*), 743
FuncComment (*class in angr.analyses.disassembly*), 742
FuncInfo (*class in angr.analyses.typehoon.typevars*), 919
FuncInfo (*class in angr.analyses.cfg.pe_msvc_eh_structs*), 342

FuncInfo (*class in angr.analyses.identifier.identify*), 784
FuncNode (*class in angr.codenode*), 988
FuncOut (*class in angr.analyses.typehoon.typevars*), 919
Function (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 798
function (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
function (*angr.analyses.reaching_definitions.FunctionCallEdge attribute*), 807
Function (*angr.analyses.reaching_definitions.subject.SubjectType attribute*), 869
FUNCTION (*angr.knowledge_plugins.CommentKind attribute*), 1124
FUNCTION (*angr.knowledge_plugins.comments.CommentKind attribute*), 1189
Function (*class in angr.analyses.typehoon.typeconsts*), 912
Function (*class in angr.knowledge_plugins.functions.function*), 1210
Function (*class in angr.protos.function_pb2*), 1402
function() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1223
function_address (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
function_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
function_address (*angr.state_plugins.InspectAttrs attribute*), 1498
function_arg (*angr.analyses.purity.engine.DataSource attribute*), 801
function_comment() (*angr.knowledge_plugins.comments.Comments method*), 1190
function_dependency_graph() (*angr.analyses.ddg.DDG method*), 365
function_final_states (*angr.analyses.vfg.VFG property*), 936
function_initial_states (*angr.analyses.vfg.VFG property*), 936
function_list_dirty (*angr.analyses.decompiler.edits.Refresh attribute*), 427
function_list_dirty (*angr.analyses.decompiler.edits.results.Refresh attribute*), 449
FUNCTION_LOCAL (*angr.analyses.flirt.consts.FlirtFunctionFlag attribute*), 755
function_name (*angr.analyses.reassembler.FunctionLabel property*), 871
function_name_changed() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1222
function_needs_variable_recovery() (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis static method*), 354
FUNCTION_PROLOGUE (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 319
function_prototype() (*angr.factory.AngrObjectFactory method*), 1113
function_summary() (*in module angr.analyses.decompiler.edits.cache*), 437
FUNCTION_UNRESOLVED_COLLISION (*angr.analyses.flirt.consts.FlirtFunctionFlag attribute*), 755
FunctionAlignmentAnalysis (*class in angr.analyses.patchfinder*), 793
FunctionAnalysis (*class in angr.analyses.vfg*), 933
FunctionCall (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 799
FunctionCallData (*class in angr.analyses.reaching_definitions.function_handler*), 839
FunctionCallDataUnwrapped (*class in angr.analyses.reaching_definitions.function_handler*), 842
FunctionCallEdge (*class in angr.analyses.cfg.cfg_fast*), 318
FunctionCallRelationships (*class in angr.analyses.reaching_definitions.function_handler*), 842
FunctionDict (*class in angr.knowledge_plugins.functions.function_manager*), 1219
FunctionDictBase (*class in angr.knowledge_plugins.functions.function_manager*), 1218
FunctionDiff (*class in angr.analyses.bindiff*), 280
FunctionEdge (*class in angr.analyses.cfg.cfg_fast*), 318
FunctionEffect (*class in angr.analyses.reaching_definitions.function_handler*), 839
FunctionFakeRetEdge (*class in angr.analyses.cfg.cfg_fast*), 319
FunctionGraphVisitor (*class in angr.analyses.forward_analysis.visitors.function_graph*), 766
FunctionHandler (*class in angr.analyses.reaching_definitions.function_handler*), 843
FunctionInfo (*class in angr.knowledge_plugins.functions.function*), 1209
FunctionKey (*class in angr.analyses.cfg.cfg_job_base*), 326
FunctionLabel (*class in angr.analyses.reassembler*), 870
FunctionLikeMacro (*class in angr.ailment.expression*), 220

FunctionManager (*class in angr.knowledge_plugins.functions.function_manager*), 1221
FunctionManagerSerializer (*class in angr.angrdb.serializers.funcs*), 951
FunctionNameSuggestion (*class in angr.llm_models*), 1295
FunctionNotFoundError, 437, 1301
FunctionNotInitialized, 770
FunctionParser (*class in angr.knowledge_plugins.functions.function_parser*), 1227
FunctionPrototypeInference (*class in angr.rust.optimization_passes.function_prototype_inference*), 1423
FunctionProxiNode (*class in angr.analyses.proximity_graph*), 799
FunctionReturn (*class in angr.analyses.cfg.cfg_fast*), 315
FunctionReturnEdge (*class in angr.analyses.cfg.cfg_fast*), 319
functions (*angr.analyses.cfg.cfg_base.CFGBase property*), 309
functions_probably_identical() (*angr.analyses.bindiff.BinDiff method*), 282
functions_reachable() (*angr.knowledge_plugins.functions.function.Function method*), 1218
FunctionStart (*class in angr.analyses.disassembly*), 740
FunctionTag (*class in angr.knowledge_plugins.key_definitions.tag*), 1261
FunctionTransitionEdge (*class in angr.analyses.cfg.cfg_fast*), 318
futex (*class in angr.procedures.linux_kernel.futex*), 1365
fwd_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
fwd_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 1560
fwrite (*class in angr.procedures.libc.fwrite*), 1356
fwrite_unlocked (*in module angr.procedures.libc.fwrite*), 1356

## G

gate (*angr.analyses.decompiler.known_patterns.KnownPatternTemplate attribute*), 463
gate (*angr.analyses.decompiler.known_patterns.templates.KnownPatternTemplate attribute*), 532
GateContext (*class in angr.analyses.decompiler.known_patterns.gating*), 511
GDB (*class in angr.state_plugins.gdb*), 1551
gen_input_output_pair() (*angr.analyses.identifier.func.Func method*), 771
gen_input_output_pair() (*angr.analyses.identifier.functions.atoi.atoi method*), 772
gen_input_output_pair() (*angr.analyses.identifier.functions.based_atoi.based_atoi method*), 772
gen_input_output_pair() (*angr.analyses.identifier.functions.fdprintf.fdprintf method*), 773
gen_input_output_pair() (*angr.analyses.identifier.functions.free.free method*), 773
gen_input_output_pair() (*angr.analyses.identifier.functions.int2str.int2str method*), 774
gen_input_output_pair() (*angr.analyses.identifier.functions.int2str.int2str_v2 method*), 774
gen_input_output_pair() (*angr.analyses.identifier.functions.int2str.int2str_v3 method*), 774
gen_input_output_pair() (*angr.analyses.identifier.functions.int2str.int2str_v4 method*), 774
gen_input_output_pair() (*angr.analyses.identifier.functions.malloc.malloc method*), 775
gen_input_output_pair() (*angr.analyses.identifier.functions.memcmp.memcmp method*), 775
gen_input_output_pair() (*angr.analyses.identifier.functions.memcpy.memcpy method*), 775
gen_input_output_pair() (*angr.analyses.identifier.functions.memset.memset method*), 776
gen_input_output_pair() (*angr.analyses.identifier.functions.printf.printf method*), 777
gen_input_output_pair() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 778
gen_input_output_pair() (*angr.analyses.identifier.functions.recv_until.receive_until_fd method*), 777
gen_input_output_pair() (*angr.analyses.identifier.functions.skip_calloc.calloc method*), 778
gen_input_output_pair() (*angr.analyses.identifier.functions.skip_realloc.realloc method*), 778

gen_input_output_pair() (*angr.analyses.identifier.functions.skip_recv_n.receive_n2 method*), 779
gen_input_output_pair() (*angr.analyses.identifier.functions.skip_recv_n.receive_n3 method*), 779
gen_input_output_pair() (*angr.analyses.identifier.functions.skip_recv_n.receive_n4 method*), 779
gen_input_output_pair() (*angr.analyses.identifier.functions.snprintf.snprintf method*), 780
gen_input_output_pair() (*angr.analyses.identifier.functions.sprintf.sprintf method*), 780
gen_input_output_pair() (*angr.analyses.identifier.functions.strcmp.strcmp method*), 782
gen_input_output_pair() (*angr.analyses.identifier.functions.strcpy.strcpy method*), 782
gen_input_output_pair() (*angr.analyses.identifier.functions.strlen.strlen method*), 783
gen_input_output_pair() (*angr.analyses.identifier.functions.strncmp.strncmp method*), 783
gen_input_output_pair() (*angr.analyses.identifier.functions.strncpy.strncpy method*), 784
gen_input_output_pair() (*angr.analyses.identifier.functions.strtol.strtol method*), 784
generate() (*angr.analyses.decompiler.known_patterns.generic_pattern_character method*), 516
generate_constraints_between() (*angr.analyses.typehoon.dfa.DFAConstraintSolver method*), 899
generate_gdt() (*angr.simos.simos.SimOS method*), 1490
generate_index() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
generate_symbolic_array() (*angr.procedures.java_lang.string.StringSplit method*), 1349
generate_symbolic_cmd_line_arg() (*angr.simos.javavm.SimJavaVM static method*), 1486
generation (*angr.state_plugins.icicle.IcicleVMRef attribute*), 1564
generic_compare() (*angr.engines.pcode.behavior.OpBehavior static method*), 1015
generic_compare() (*angr.engines.vex.claripy.irop.SimIROp method*), 1074
generic_info_hook() (*in module angr.state_plugins.trace_additions*), 1597
generic_minmax() (*angr.engines.vex.claripy.irop.SimIROp method*), 1074
generic_rotate_with_carry() (*in module angr.engines.vex.claripy.ccall*), 1071
generic_shift_thing() (*angr.engines.vex.claripy.irop.SimIROp method*), 1074
get() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 895
get() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method*), 1179
get() (*angr.knowledge_plugins.functions.function_manager.FunctionDictB method*), 1218
get() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 1247
get() (*angr.knowledge_plugins.labels.Labels method*), 1264
get() (*angr.knowledge_plugins.propagations.propagation_manager.Propag method*), 1271
get() (*angr.knowledge_plugins.structured_code.StructuredCodeManager method*), 1279
get() (*angr.procedures.definitions.SimCppLibrary method*), 1328
get() (*angr.procedures.definitions.SimLibrary method*), 1327
get() (*angr.procedures.definitions.SimSyscallLibrary method*), 1331
get() (*angr.procedures.definitions.SimTypeCollection method*), 1324
get() (*angr.rust.knowledge_plugins.clinic_factory.ClinicFactory method*), 1412
get() (*angr.rust.knowledge_plugins.known_structs.KnownStructs method*), 1412
get() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
get() (*angr.state_plugins.filesystem.SimMount method*), 1549
get() (*angr.state_plugins.globals.SimStateGlobals method*), 1551
get() (*angr.state_plugins.view.SimRegNameView method*), 1604
get() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitma method*), 1652
get() (*angr.utils.mp.Initializer class method*), 1694
get_abstract_locations() (*angr.storage.memory_mixins.regioned_memory.region_meta_mi method*), 1662
get_addr() (*angr.annocfg.AnnotatedCFG method*), 957
get_addr_of_native_method() (*angr.simos.javavm.SimJavaVM method*), 1487
get_addrs_by_name()

(*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1222
get_all_definitions() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 886
get_all_definitions() (*in module angr.analyses.reaching_definitions*), 833
get_all_funcaddrs_from_signature() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1226
get_all_nodes() (*angr.analyses.vfg.VFG method*), 936
get_all_nodes() (*angr.analyses.vsa_ddg.VSA_DDG method*), 937
get_all_nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1168
get_all_nodes_intersecting_region() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1168
get_all_patches() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
get_all_predecessors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1171
get_all_successors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1171
get_all_tmp_definitions() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 887
get_all_variables() (*angr.keyed_region.KeyedRegion method*), 1118
get_alloc_depth() (*angr.state_plugins.uc_manager.SimUCManager method*), 1598
get_angr_type_from_name() (*in module angr.procedures.definitions.parse_win32json*), 1334
get_any_node() (*angr.analyses.vfg.VFG method*), 936
get_any_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1168
get_arg_info() (*angr.calling_conventions.SimCC method*), 971
get_args() (*angr.calling_conventions.SimCC method*), 969
get_ast_subexprs() (*in module angr.analyses.decompiler.utils*), 721
get_base_addr() (*angr.keyed_region.KeyedRegion method*), 1118
get_base_call_state() (*angr.analyses.identifier.runner.Runner method*), 785
get_base_state() (*angr.analyses.identifier.custom_callable.IdentifierCallable method*), 770
get_basic_blocks() (*in module angr.mcp.server*), 1304
get_basic_info() (*in module angr.flirt.build_sig*), 1114
get_behavior_for_opcode() (*angr.engines.pcode.behavior.BehaviorFactory method*), 1034
get_binary_name_cached() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1221
get_bits() (*in module angr.ailment.utils*), 227
get_block() (*angr.knowledge_plugins.functions.function.Function method*), 1212
get_block_key() (*in module angr.knowledge_plugins.cfg.spilling_cfg*), 1181
get_block_size() (*angr.knowledge_plugins.functions.function.Function method*), 1212
get_bookmark() (*angr.knowledge_plugins.bookmarks.Bookmarks method*), 1149
get_branching_nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1171
get_by_addr() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1222
get_by_name() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1222
get_byte() (*angr.state_plugins.trace_additions.ChallRespInfo static method*), 1597
get_cache() (*in module angr.analyses.decompiler.edits.cache*), 435
get_call_args() (*angr.analyses.identifier.identify.Identifier method*), 785
get_call_clobbered_regs() (*in module angr.analyses.s_reaching_definitions.s_rda_view*), 888
get_call_return() (*angr.knowledge_plugins.functions.function.Function method*), 1215
get_call_sites() (*angr.knowledge_plugins.functions.function.Function method*), 1215
get_call_stack_suffix() (*angr.analyses.cfg.cfg_job_base.CFGJobBase method*), 326
get_call_target() (*angr.knowledge_plugins.functions.function.Function method*), 1215
get_call_target() (*angr.rust.mixins.cfa_mixin.CFAMixin method*), 1417
get_callgraph() (*in module angr.mcp.server*), 1306
get_cc() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 1151
get_cfg() (*in module angr.mcp.server*), 1303
get_cfg_edge_cache_limit() (*angr.project.Project method*), 1393
get_cfg_node_cache_limit() (*angr.project.Project method*), 1393

get_class() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 1573
get_class_hierarchy() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 1573
get_cmd_line_args() (*angr.simos.javavm.SimJavaVM static method*), 1487
get_concrete_fd() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
get_concrete_state() (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFast method*), 925
get_concrete_value() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 866
get_concrete_value() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1254
get_cpp_function_name() (*in module angr.utils.library*), 1692
get_cpp_function_name_and_metadata() (*in module angr.utils.library*), 1692
get_data_dep() (*angr.analyses.data_dep.data_dependency_analysis.DataDependencyAnalysis method*), 358
get_data_references() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 1174
get_data_regions_from_meta_regions() (*in module angr.analyses.cfg.meta_structs*), 341
get_data_size() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
get_dbinfo() (*angr.angrdb.db.AngrDB method*), 941
get_def_block_and_stmt() (*angr.rust.mixins.dfa_mixin.DFAMixin method*), 1419
get_def_by_vvar() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1420
get_default_cache_limit() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1221
get_default_sig_dir() (*in module angr.rust.utils.rust_sigs*), 1439
get_default_value() (*angr.engines.soot.values.arrayref.SimSootValue_ArrayBaseRef method*), 1059
get_default_value_by_type() (*angr.simos.javavm.SimJavaVM static method*), 1487
get_definition_by_varid() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 886
get_definitions() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 865
get_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1253
get_defs() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 1259
get_defs_by_location() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 887
get_dependants() (*angr.analyses.cdg.CDG method*), 290
get_edge_data() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1184
get_element() (*angr.analyses.decompiler.structured_codegen.base.PositionMapping method*), 668
get_existing_vvar() (*angr.sim_state.SimState method*), 1430
get_exit_livedefinitions() (*in module angr.analyses.reaching_definitions.function_handler*), 838
get_exit_stmt_idx() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1171
get_exports() (*in module angr.mcp.server*), 1306
get_expr_depth() (*angr.state_plugins.trace_additions.ZenPlugin method*), 1598
get_expr_shift_left_amount() (*in module angr.analyses.decompiler.peephole_optimizations.utils*), 621
get_fd() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
get_field() (*angr.engines.soot.values.thisref.SimSootValue_ThisRef method*), 1062
get_field_offset() (*angr.rust.sim_type.RustSimStruct method*), 1433
get_field_ty() (*angr.rust.sim_type.RustSimStruct method*), 1433
get_flag_bytes() (*angr.state_plugins.trace_additions.ZenPlugin method*), 1598
get_flag_rand_args() (*angr.state_plugins.trace_additions.ZenPlugin static method*), 1598
get_footprint() (*angr.calling_conventions.SimFunctionArgument method*), 964
get_from_signature() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1225
get_func_addr_with_block_count() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1225
get_func_block_count() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1225
get_func_info() (*angr.analyses.identifier.identify.Identifier method*), 785
get_function() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1225

*method*), 1225
get_function_cache_limit() (*angr.project.Project method*), 1392
get_function_diff() (*angr.analyses.bindiff.BinDiff method*), 282
get_function_hints_from_meta_regions() (*in module angr.analyses.cfg.meta_structs*), 341
get_function_info() (*in module angr.mcp.server*), 1304
get_function_manager() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1289
get_function_name() (*in module angr.utils.library*), 1691
get_function_subgraph() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 314
get_global_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1286
get_guardians() (*angr.analyses.cdg.CDG method*), 290
get_heap_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1254
get_heap_offset() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState static method*), 861
get_heap_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1257
get_implementers() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
get_imports() (*in module angr.mcp.server*), 1306
get_info() (*angr.angrdb.db.AngrDB static method*), 941
get_interface() (*angr.analyses.outliner.outliner.Outliner method*), 792
get_intersecting_functions() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1172
get_irsb_at() (*angr.analyses.datagraph_meta.DataGraphMeta method*), 361
get_kb() (*angr.project.Project method*), 1390
get_key_func_addrs() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1226
get_key_func_type_and_addrs() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1226
get_knowledge() (*angr.knowledge_base.KnowledgeBase method*), 1119
get_last_statement() (*angr.analyses.decompiler.condition_processor.ConditionProcessor class method*), 409
get_last_statement_index() (*angr.annocfg.AnnotatedCFG method*), 957
get_last_statements() (*angr.analyses.decompiler.condition_processor.ConditionProcessor class method*), 409
get_loop_back_edges() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
get_loops() (*angr.annocfg.AnnotatedCFG method*), 958
get_map_key() (*in module angr.procedures.java_util.map*), 1350
get_max_sinkhole() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
get_memory_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1254
get_model() (*angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager method*), 1249
get_most_accurate() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 1165
get_mountpoint() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
get_name() (*angr.analyses.identifier.func.Func method*), 771
get_name() (*angr.analyses.identifier.functions.atoi.atoi method*), 772
get_name() (*angr.analyses.identifier.functions.based_atoi.based_atoi method*), 772
get_name() (*angr.analyses.identifier.functions.fdprintf.fdprintf method*), 773
get_name() (*angr.analyses.identifier.functions.free.free method*), 773
get_name() (*angr.analyses.identifier.functions.int2str.int2str method*), 774
get_name() (*angr.analyses.identifier.functions.int2str.int2str_v2 method*), 774
get_name() (*angr.analyses.identifier.functions.int2str.int2str_v3 method*), 774
get_name() (*angr.analyses.identifier.functions.int2str.int2str_v4 method*), 774
get_name() (*angr.analyses.identifier.functions.malloc.malloc method*), 775
get_name() (*angr.analyses.identifier.functions.memcmp.memcmp method*), 775
get_name() (*angr.analyses.identifier.functions.memcpy.memcpy method*), 775
get_name() (*angr.analyses.identifier.functions.memset.memset method*), 776
get_name() (*angr.analyses.identifier.functions.printf.printf method*), 777
get_name() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 778
get_name() (*angr.analyses.identifier.functions.recv_until_fd

*method*), 777
get_name() (*angr.analyses.identifier.functions.skip_calloc.calloc method*), 778
get_name() (*angr.analyses.identifier.functions.skip_realloc.realloc method*), 778
get_name() (*angr.analyses.identifier.functions.skip_recv_n.receive_n method*), 779
get_name() (*angr.analyses.identifier.functions.skip_recv_n.receive_n method*), 779
get_name() (*angr.analyses.identifier.functions.skip_recv_n.receive_n method*), 778
get_name() (*angr.analyses.identifier.functions.snprintf.snprintf method*), 780
get_name() (*angr.analyses.identifier.functions.sprintf.sprintf method*), 780
get_name() (*angr.analyses.identifier.functions.strcasecmp.strcasecmp method*), 781
get_name() (*angr.analyses.identifier.functions.strcmp.strcmp method*), 781
get_name() (*angr.analyses.identifier.functions.strcpy.strcpy method*), 782
get_name() (*angr.analyses.identifier.functions.strlen.strlen method*), 783
get_name() (*angr.analyses.identifier.functions.strncmp.strncmp method*), 783
get_name() (*angr.analyses.identifier.functions.strncpy.strncpy method*), 784
get_name() (*angr.analyses.identifier.functions.strtol.strtol method*), 784
get_native_cc() (*angr.simos.javavm.SimJavaVM method*), 1487
get_native_type() (*angr.simos.javavm.SimJavaVM method*), 1487
get_nearest_pos() (*angr.analyses.decompiler.structured_codegen.base.PositionMapping method*), 669
get_new_uuid() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin static method*), 1636
get_next_xref_addr_by_dst() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
get_node() (*angr.analyses.decompiler.structured_codegen.base.PositionMapping method*), 668
get_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1167
get_node() (*angr.knowledge_plugins.functions.function.Function method*), 1212
get_node_by_key() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1182
get_node_order() (*angr.utils.graph.DirectedGraphHelper method*), 1689
get_normalized_block() (*angr.analyses.bindiff.FunctionDiff static method*), 281
get_objects_by_offset() (*angr.keyed_region.KeyedRegion method*), 1118
get_observation_by_exit() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefi method*), 1260
get_observation_by_insn() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefi method*), 1260
get_observation_by_node() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefi method*), 1260
get_observation_by_stmt() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefi method*), 1260
get_one_predecessor() (*angr.rust.mixins.cfa_mixin.CFAMixin method*), 1417
get_one_successor() (*angr.rust.mixins.cfa_mixin.CFAMixin method*), 1417
get_one_value() (*angr.analyses.reaching_definitions.rd_state.ReachingD method*), 866
get_one_value() (*angr.knowledge_plugins.key_definitions.live_definition method*), 1254
get_optimization_passes() (*angr.analyses.decompiler.presets.preset.DecompilationPreset method*), 622
get_optimization_passes() (*in module angr.analyses.decompiler.optimization_passes*), 558
get_out_state() (*angr.analyses.identifier.runner.Runner method*), 785
get_own() (*angr.knowledge_plugins.types.TypesStore method*), 1280
get_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 1275
get_phi_subvariables() (*angr.knowledge_plugins.variables.variable_manager.VariableM method*), 1286
get_phi_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableM method*), 1286
get_plugin() (*angr.knowledge_base.KnowledgeBase method*), 1119
get_plugin() (*angr.misc.plugins.PluginHub method*), 1318
get_pointer_array_hints() (*in module angr.analyses.cfg.meta_structs*), 341
get_pointer_array_hints_pe() (*in module angr.analyses.cfg.meta_structs*), 341
get_possible_len() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 1597
get_post_dominators() (*angr.analyses.cdg.CDG method*), 290

get_predecessors() (*angr.analyses.ddg.DDG method*), 365
get_predecessors() (*angr.analyses.vsa_ddg.VSA_DDG method*), 937
get_predecessors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1169
get_predecessors_and_jumpkind() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1170
get_predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1170
get_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 1151
get_prototype() (*angr.procedures.definitions.SimCppLibrary method*), 1329
get_prototype() (*angr.procedures.definitions.SimLibrary method*), 1327
get_prototype() (*angr.procedures.definitions.SimSyscallLibrary method*), 1332
get_reaching_definitions_by_insn() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 868
get_reaching_definitions_by_node() (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis method*), 868
get_real_len() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 1597
get_recent_bbl_addrs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
get_recent_n() (*angr.exploration_techniques.spiller.PickleStashSpiller method*), 1105
get_ref() (*angr.engines.soot.values.instancefieldref.SimSootValue_InstanceFieldRef class method*), 1060
get_ref() (*angr.engines.soot.values.staticfieldref.SimSootValue_StaticFieldRef class method*), 1061
get_ref() (*angr.state_hierarchy.StateHierarchy method*), 1494
get_reg_name() (*angr.analyses.identifier.identify.Identifier static method*), 785
get_reg_vvar_by_insn() (*angr.analyses.s_reaching_definitions.s_rda_view.SRDAView method*), 890
get_reg_vvar_by_stmt() (*angr.analyses.s_reaching_definitions.s_rda_view.SRDAView method*), 889
get_register_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1254
get_regs() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
get_run() (*angr.annocfg.AnnotatedCFG method*), 957
get_rust_function_name() (*in module angr.utils.library*), 1692
get_rust_optimization_passes() (*in module angr.rust.optimization_passes*), 1421
get_same_length_constraints() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 1597
get_segdescr_base() (*in module angr.engines.vex.claripy.ccall*), 1072
get_segdescr_limit() (*in module angr.engines.vex.claripy.ccall*), 1072
get_segment_register_name() (*angr.simos.linux.SimLinux method*), 1488
get_segment_register_name() (*angr.simos.windows.SimWindows method*), 1493
get_session() (*angr.mcp.session.SessionManager method*), 1308
get_session_manager() (*in module angr.mcp.session*), 1309
get_single_footprint() (*angr.calling_conventions.SimStructArg method*), 865
get_size() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
get_smc_addrs_and_lengths() (*angr.analyses.smc.TraceClassifier method*), 892
get_sp() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitions method*), 862
get_sp() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1252
get_sp_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1252
get_stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitions method*), 862
get_stack_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1252
get_stack_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1254
get_stack_offset() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitions method*), 861
get_stack_offset() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase method*), 928
get_stack_offset() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1251
get_stack_values() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1254
get_stack_vvar_and_offset_by_insn() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1420
get_stack_vvar_by_insn()

(*angr.analyses.s_reaching_definitions.s_rda_view.SRDAView method*), 890
get_stack_vvar_by_insn() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1420
get_stack_vvar_by_stmt() (*angr.analyses.s_reaching_definitions.s_rda_view.SRDAView method*), 890
get_stackarg_insaddrs() (*angr.analyses.decompiler.stackarg_offset_manager.StackArgOffsetManager method*), 660
get_stackarg_offsets() (*angr.analyses.decompiler.stackarg_offset_manager.StackArgOffsetManager method*), 660
get_stackvar_max_sizes() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1288
get_stdin_indices() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 1597
get_stdout_indices() (*angr.state_plugins.trace_additions.ChallRespInfo method*), 1597
get_stop_details() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
get_stop_msg() (*angr.state_plugins.unicorn_engine.STOP static method*), 1602
get_strings() (*in module angr.mcp.server*), 1305
get_structurer_option() (*in module angr.analyses.decompiler.decompilation_options*), 414
get_stub() (*angr.procedures.definitions.SimCppLibrary method*), 1329
get_stub() (*angr.procedures.definitions.SimLibrary method*), 1327
get_stub() (*angr.procedures.definitions.SimSyscallLibrary method*), 1331
get_sub_classes() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
get_sub_classes_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
get_sub_interfaces() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
get_sub_interfaces_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
get_subgraph() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 314
get_successors() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1169
get_successors_and_jumpkind() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1170
get_successors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1170
get_super_classes() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
get_super_classes_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
get_superclass() (*angr.state_plugins.javavm_classloader.SimJavaVmCl method*), 1573
get_symbolic_addrs() (*angr.storage.memory_mixins.convenient_mappings_mixin.Conve method*), 1634
get_variable() (*angr.annotatedcfg.AnnotatedCFG method*), 958
get_terminal_call() (*in module angr.rust.utils.ail*), 1438
get_terminal_vvar() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1420
get_terminal_vvar_value() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1420
get_terminal_vvar_values() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1420
get_tmp_def_stmt() (*in module angr.utils.vex*), 1703
get_tmp_definitions() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefi method*), 1253
get_tmp_deflocs() (*in module angr.utils.ssa*), 1696
get_tmp_uselocs() (*in module angr.utils.ssa*), 1696
get_tmp_uses() (*angr.analyses.s_reaching_definitions.s_rda_model.SRD method*), 888
get_topological_order() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 314
get_tracer() (*in module angr.misc.telemetry*), 1321
get_tv_max_stack_size() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 903
get_type() (*angr.state_plugins.trace_additions.FormatInfo method*), 1596
get_type() (*angr.state_plugins.trace_additions.FormatInfoDontConstrain method*), 1597
get_type() (*angr.state_plugins.trace_additions.FormatInfoIntToStr method*), 1597
get_type() (*angr.state_plugins.trace_additions.FormatInfoStrToInt method*), 1596
get_type_hints() (*angr.rust.knowledge_plugins.type_hints.TypeHints method*), 1413
get_type_variable()

(*angr.analyses.typehoon.typevars.TypeVariables method*), 918
get_typeref_if_available() (*in module angr.procedures.definitions.parse_win32json*), 1334
get_unambiguous_name() (*angr.knowledge_plugins.functions.function.Function method*), 1217
get_unconstrained_simprocedure() (*angr.engines.soot.engine.SootMixin method*), 1047
get_unified_local_vars() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 673
get_unified_local_vars() (*angr.analyses.decompiler.structured_codegen.rust.RustFunction method*), 693
get_unified_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1285
get_unique_label() (*angr.knowledge_plugins.labels.Labels method*), 1264
get_unique_strings() (*in module angr.flirt.build_sig*), 1115
get_unique_symbol_name() (*angr.analyses.reassembler.SymbolManager method*), 871
get_uses() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1262
get_uses_by_def() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 888
get_uses_by_insaddr() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1263
get_uses_by_location() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 887
get_uses_by_location() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1263
get_uses_with_expr() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1262
get_value() (*angr.calling_conventions.SimArrayArg method*), 966
get_value() (*angr.calling_conventions.SimComboArg method*), 965
get_value() (*angr.calling_conventions.SimFunctionArgument method*), 964
get_value() (*angr.calling_conventions.SimLyingRegArg method*), 971
get_value() (*angr.calling_conventions.SimReferenceArgument method*), 966
get_value() (*angr.calling_conventions.SimRegArg method*), 964
get_value() (*angr.calling_conventions.SimStackArg method*), 965
get_value() (*angr.calling_conventions.SimStructArg method*), 966
get_values() (*angr.analyses.reaching_definitions.rd_state.ReachingDefin method*), 865
get_values() (*angr.knowledge_plugins.key_definitions.live_definitions.Li method*), 1254
get_variable() (*angr.analyses.decompiler.optimization_passes.engine_b method*), 575
get_variable() (*angr.state_plugins.debug_variables.SimDebugVariableI method*), 1546
get_variable_accesses() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 1289
get_variable_accesses() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 1285
get_variable_definitions() (*angr.analyses.variable_recovery.variable_recovery_base.Variabl method*), 927
get_variable_definitions() (*angr.analyses.variable_recovery.variable_recovery_base.Variabl method*), 929
get_variable_type() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 1287
get_variables() (*angr.knowledge_plugins.variables.variable_manager.V method*), 1285
get_variables() (*angr.state_plugins.solver.SimSolver method*), 1588
get_variables_by_offset() (*angr.keyed_region.KeyedRegion method*), 1118
get_variables_without_writes() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 1286
get_variant() (*angr.analyses.typehoon.typeconsts.RustEnum method*), 911
get_variant() (*angr.rust.sim_type.RustSimEnum method*), 1435
get_variant_by_name() (*angr.rust.sim_type.RustSimEnum method*), 1435
get_venv() (*in module angr.misc.bug_report*), 1315
get_vvar_deflocs() (*in module angr.utils.ssa*), 1696
get_vvar_type() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1420
get_vvar_uselocs() (*in module angr.utils.ssa*), 1696
get_vvar_uses() (*angr.analyses.s_reaching_definitions.s_rda_model.SRI method*), 887
get_vvar_uses_with_expr() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel*

*method*), 887
get_vvar_value() (*angr.analyses.s_reaching_definitions.s_rda_view.SRDAView method*), 891
get_vvar_value() (*angr.rust.mixins.srda_mixin.SRDAMixin method*), 1419
get_whitelisted_statements() (*angr.annocfg.AnnotatedCFG method*), 957
get_xrefs() (*in module angr.mcp.server*), 1305
get_xrefs_by_dst() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
get_xrefs_by_dst_region() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
get_xrefs_by_ins_addr() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
get_xrefs_by_ins_addr_region() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1293
get_xrefs_in_range() (*angr.knowledge_plugins.xrefs.xref_manager.XRefManager method*), 1292
GetArrayElements (*class in angr.procedures.java_jni.array_operations*), 1339
GetArrayLength (*class in angr.procedures.java_jni.array_operations*), 1338
GetArrayRegion (*class in angr.procedures.java_jni.array_operations*), 1339
GetBitsTypeParams (*in module angr.ailment.utils*), 227
GetBooleanField (*class in angr.procedures.java_jni.field_access*), 1341
GetByteField (*class in angr.procedures.java_jni.field_access*), 1341
getc (*in module angr.procedures.libc.fgetc*), 1354
getc_unlocked (*in module angr.procedures.libc.fgetc*), 1354
getchar (*class in angr.procedures.libc.getchar*), 1356
getchar_unlocked (*in module angr.procedures.libc.getchar*), 1356
GetCharField (*class in angr.procedures.java_jni.field_access*), 1341
GetCommandLineA (*class in angr.procedures.win32.GetCommandLine*), 1382
GetCommandLineW (*class in angr.procedures.win32.GetCommandLine*), 1382
GetCurrentProcessId (*class in angr.procedures.win32.GetCurrentProcessId*), 1382
GetCurrentThreadId (*class in angr.procedures.win32.GetCurrentThreadId*), 1382
getcwd (*class in angr.procedures.linux_kernel.cwd*), 1365
getegid (*class in angr.procedures.libc.getegid*), 1356
getegid (*class in angr.procedures.linux_kernel.getegid*), 1365
getegid32 (*class in angr.procedures.linux_kernel.getegid*), 1365
getenv (*class in angr.procedures.posix.getenv*), 1372
geteuid (*class in angr.procedures.libc.geteuid*), 1356
geteuid (*class in angr.procedures.linux_kernel.geteuid*), 1365
geteuid32 (*class in angr.procedures.linux_kernel.geteuid*), 1365
GetField (*class in angr.procedures.java_jni.field_access*), 1341
GetFieldID (*class in angr.procedures.java_jni.field_access*), 1340
getgid (*class in angr.procedures.libc.getgid*), 1356
getgid (*class in angr.procedures.linux_kernel.getgid*), 1365
getgid (*class in angr.procedures.linux_kernel.uid*), 1367
getgid32 (*class in angr.procedures.linux_kernel.getgid*), 1365
gethostbyname (*class in angr.procedures.posix.gethostbyname*), 1372
gethostbyname (*class in angr.procedures.win32.gethostbyname*), 1384
GetIntField (*class in angr.procedures.java_jni.field_access*), 1341
GetIterator (*class in angr.procedures.java_util.collection*), 1350
GetKeyboardType (*class in angr.procedures.win_user32.keyboard*), 1387
GetLastInputInfo (*class in angr.procedures.win32.GetLastInputInfo*), 1382
GetLocalTime (*class in angr.procedures.win32.sim_time*), 1386
GetLongField (*class in angr.procedures.java_jni.field_access*), 1341
GetMethodID (*class in angr.procedures.java_jni.method_calls*), 1342
GetModuleHandleA (*class in angr.procedures.win32.GetModuleHandle*), 1382
GetModuleHandleW (*class in angr.procedures.win32.GetModuleHandle*), 1382
GetObjectArrayElement (*class in angr.procedures.java_jni.array_operations*), 1339

GetObjectClass (*class in angr.procedures.java_jni.object_operations*), 1347
GetObjectField (*class in angr.procedures.java_jni.field_access*), 1341
getpass (*class in angr.procedures.posix.getpass*), 1372
getpid (*class in angr.procedures.linux_kernel.getpid*), 1365
getpiece() (*angr.analyses.disassembly.DisassemblyPiece method*), 739
getppid (*class in angr.procedures.linux_kernel.getpid*), 1365
GetProcAddress (*class in angr.procedures.win32.dynamic_loading*), 1384
GetProcessAffinityMask (*class in angr.procedures.win32.GetProcessAffinityMask*), 1383
GetProcessHeap (*class in angr.procedures.win32.heap*), 1384
getresgid (*class in angr.procedures.linux_kernel.uid*), 1367
getresuid (*class in angr.procedures.linux_kernel.uid*), 1368
getrlimit (*class in angr.procedures.linux_kernel.getrlimit*), 1366
gets (*class in angr.procedures.libc.gets*), 1356
GetShortField (*class in angr.procedures.java_jni.field_access*), 1341
GetSimpleName (*class in angr.procedures.java_lang.getsimplename*), 1348
getsockopt (*class in angr.procedures.posix.getsockopt*), 1372
getstate() (*angr.calling_conventions.ArgSession method*), 966
getstate() (*angr.calling_conventions.SerializableCounter method*), 963
getstate() (*angr.calling_conventions.SerializableIterator method*), 963
getstate() (*angr.calling_conventions.SerializableListIterator method*), 963
getstate() (*angr.calling_conventions.SimCC.ArgSession method*), 968
getstate() (*angr.calling_conventions.UsercallArgSession method*), 967
GetStaticBooleanField (*class in angr.procedures.java_jni.field_access*), 1340
GetStaticByteField (*class in angr.procedures.java_jni.field_access*), 1340
GetStaticCharField (*class in angr.procedures.java_jni.field_access*), 1340
GetStaticField (*class in angr.procedures.java_jni.field_access*), 1340
GetStaticIntField (*class in angr.procedures.java_jni.field_access*), 1340
GetStaticLongField (*class in angr.procedures.java_jni.field_access*), 1340
GetStaticObjectField (*class in angr.procedures.java_jni.field_access*), 1340
GetStaticShortField (*class in angr.procedures.java_jni.field_access*), 1340
GetStdHandle (*class in angr.procedures.win32.file_handles*), 1384
GetStringUTFChars (*class in angr.procedures.java_jni.string_operations*), 1347
GetStringUTFLength (*class in angr.procedures.java_jni.string_operations*), 1347
GetSuperclass (*class in angr.procedures.java_jni.class_and_interface_operations*), 1340
GetSystemTimeAsFileTime (*class in angr.procedures.win32.sim_time*), 1386
GetTempPathA (*class in angr.procedures.win32.system_paths*), 1387
GetTickCount (*class in angr.procedures.win32.sim_time*), 1386
GetTickCount64 (*class in angr.procedures.win32.sim_time*), 1386
gettid (*class in angr.procedures.linux_kernel.gettid*), 1366
gettimeofday (*class in angr.procedures.posix.sim_time*), 1375
getuid (*class in angr.procedures.libc.getuid*), 1356
getuid (*class in angr.procedures.linux_kernel.getuid*), 1366
getuid (*class in angr.procedures.linux_kernel.uid*), 1367
getuid32 (*class in angr.procedures.linux_kernel.getuid*), 1366
GetVersion (*class in angr.procedures.java_jni.version_information*), 1347
GetWindowsDirectoryA (*class in angr.procedures.win32.system_paths*), 1387
give_up_on_memory_tracking() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 895
global_addr (*angr.analyses.decompiler.edits.resolve.ResolvedVariable attribute*), 447
global_addr (*angr.analyses.decompiler.edits.ResolvedVariable attribute*), 428
global_addr (*angr.knowledge_plugins.key_definitions.definition.Definitio attribute*), 1246
global_variable_at() (*in module angr.analyses.decompiler.edits.ops*), 445
GlobalAlloc (*class in angr.procedures.win32.heap*), 1384

GlobalDescriptorTable (*class in angr.simos.simos*), 1490
GO_NORETURN_NAMES (*in module angr.utils.go_runtime*), 1681
go_preamble_supported() (*in module angr.analyses.cfg.go_prologue*), 326
Goto (*class in angr.analyses.decompiler.goto_manager*), 452
GotoManager (*class in angr.analyses.decompiler.goto_manager*), 452
gotos_in_block() (*angr.analyses.decompiler.goto_manager.GotoManager method*), 452
GotoSimplifier (*class in angr.analyses.decompiler.region_simplifiers.goto*), 637
GOTPLTEntry (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
GOTPLTEntry (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
GOTPLTEntry (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
gp_register_read_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHook method*), 337
gp_register_write_hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.MIPSGPHook method*), 337
graph (*angr.analyses.cdg.CDG property*), 290
graph (*angr.analyses.cfg.cfg_base.CFGBase property*), 309
graph (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 314
graph (*angr.analyses.cfg.cfg_fast.CFGFast property*), 324
graph (*angr.analyses.data_dep.data_dependency_analysis.DataDependencyGraphAnalysis property*), 358
graph (*angr.analyses.ddg.DDG property*), 364
graph (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 627
graph (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis property*), 764
graph (*angr.analyses.reaching_definitions.dep_graph.DepGraph property*), 835
graph (*angr.knowledge_plugins.functions.function.Function property*), 1215
graph_add_edge() (*angr.analyses.decompiler.region_overlay.OverlayManager method*), 625
graph_ex() (*angr.knowledge_plugins.functions.function.Function method*), 1215
graph_remove_edge() (*angr.analyses.decompiler.region_overlay.OverlayManager method*), 625
graph_strip_entry_tags() (*angr.analyses.decompiler.region_overlay.OverlayManager method*), 625
graph_to_epsilon_nfa() (*angr.analyses.typehoon.dfa.DFAConstraintSolver static method*), 899
graph_updated() (*angr.utils.doms.IncrementalDominators method*), 1677
graph_with_successors (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 627
GraphChangedNotification, 717
GraphDephication (*class in angr.analyses.decompiler.dephication.graph_dephication*), 419
GraphDephicationVVarMapping (*class in angr.analyses.decompiler.dephication.graph_vvar_mapping*), 419
GraphRewritingAnalysis (*class in angr.analyses.decompiler.dephication.graph_rewriting*), 419
GraphUtils (*class in angr.utils.graph*), 1685
GraphVisitor (*class in angr.analyses.forward_analysis.visitors.graph*), 748
green (*angr.misc.ansi.BackgroundColor attribute*), 1314
green (*angr.misc.ansi.Color attribute*), 1314
GT (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.C attribute*), 639
GUARD (*angr.analyses.reaching_definitions.AtomKind attribute*), 806
guard (*angr.errors.SimError attribute*), 1083
GUARD (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 1241
guard (*angr.SimError attribute*), 177
GuardUse (*class in angr.knowledge_plugins.key_definitions.atoms*), 1243
guess_prototype() (*angr.calling_conventions.SimCC static method*), 969
guess_value_type() (*in module angr.analyses.decompiler.structured_codegen.c*), 671
guess_value_type() (*in module angr.analyses.decompiler.structured_codegen.rust*), 692
GUESSED (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 1209
GUESSED (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
guessed_cc (*angr.analyses.reaching_definitions.function_handler.Function attribute*), 840
guessed_cc (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
guessed_prototype (*angr.analyses.reaching_definitions.function_handler attribute*), 840

guessed_prototype (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807

## H

handle() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 708
handle() (*angr.analyses.deobfuscator.scope_ops_analyzer.ScopeOpsWalker method*), 738
handle() (*angr.procedures.win32.GetModuleHandle.GetModuleHandleA method*), 1382
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsImplicit method*), 687
handle_CAssignment() (*angr.analyses.decompiler.structured_codegen.c.PointerArithmeticFixer method*), 688
handle_CAssignment() (*angr.analyses.deobfuscator.api_obf_finder.Type1AssignmentFinder method*), 735
handle_CAssignment() (*angr.analyses.loop_analysis.loop_analysis.AssignmentToASTVisitor method*), 788
handle_CAssignment() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyAssignmentCollector method*), 789
handle_CAssignment() (*angr.analyses.loop_analysis.loop_analysis.LoopVisitor method*), 790
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 687
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsImplicit method*), 687
handle_CBinaryOp() (*angr.analyses.decompiler.structured_codegen.c.PointerArithmeticFixer method*), 688
handle_CBinaryOp() (*angr.analyses.deobfuscator.scope_ops_analyzer.ScopeOpsWalker method*), 738
handle_CBinaryOp() (*angr.analyses.loop_analysis.loop_analysis.AssignmentToASTVisitor method*), 789
handle_CBreak() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyControlStatementCollector method*), 789
handle_CConstant() (*angr.analyses.loop_analysis.loop_analysis.AssignmentToASTVisitor method*), 789
handle_CContinue() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyControlStatementCollector method*), 789
handle_CDoWhileLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CDoWhileLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyAssignmentCollector method*), 790
handle_CDoWhileLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyControlStatementCollector method*), 789
handle_CDoWhileLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopVisitor method*), 790
handle_CExpressionStatement() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CForLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CForLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyAssignmentCollector method*), 790
handle_CForLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyControlStatementCollector method*), 789
handle_CForLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopVisitor method*), 790
handle_CFunction() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CFunctionCall() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CFunctionCall() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsImplicit method*), 687
handle_CGoto() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CGoto() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyAssignmentCollector method*), 789
handle_CIfBreak() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CIfElse() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CIfElse() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyAssignmentCollector method*), 789
handle_CIfElse() (*angr.analyses.loop_analysis.loop_analysis.LoopVisitor method*), 790
handle_CIndexedVariable() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CITE() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 687
handle_CReturn() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CReturn() (*angr.analyses.decompiler.structured_codegen.c.MakeTypecastsImplicit method*), 687
handle_CStatements() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CStatements() (*angr.analyses.loop_analysis.loop_analysis.LoopVisitor method*), 790
handle_CSwitchCase()

(*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CSwitchCase() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyAssignmentCollector method*), 790
handle_CSwitchCase() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyControlStatementCollector method*), 789
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 687
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.FieldReference method*), 687
handle_CTypeCast() (*angr.analyses.decompiler.structured_codegen.c.Mod_TypecastsImplicit method*), 687
handle_CUnaryOp() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 687
handle_CUnaryOp() (*angr.analyses.deobfuscator.scope_opsanalyzer.ScopeOpsWalker method*), 738
handle_CVariable() (*angr.analyses.loop_analysis.loop_analysis.AssignmentToASTVisitor method*), 788
handle_CVariableField() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CVectorConvert() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 687
handle_CVEXCCallExpression() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 687
handle_CWhileLoop() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_CWhileLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyAssignmentCollector method*), 790
handle_CWhileLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopBodyControlStatementCollector method*), 789
handle_CWhileLoop() (*angr.analyses.loop_analysis.loop_analysis.LoopVisitor method*), 790
handle_default() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeWalker method*), 686
handle_default() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 708
handle_exception() (*angr.simos.simos.SimOS method*), 1489
handle_external_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 844
handle_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 843
handle_generic_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 844
handle_indirect_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHa method*), 844
handle_json_type() (*in module angr.procedures.definitions.parse_win32json*), 1334
handle_local_function() (*angr.analyses.find_objects_static.NewFunctionHandler method*), 746
handle_local_function() (*angr.analyses.reaching_definitions.function_handler.FunctionHa method*), 844
handle_pcode_block() (*angr.engines.pcode.emulate.PcodeEmulatorMixin method*), 1036
handle_printf() (*in module angr.analyses.reaching_definitions.function_handler_library.stdio*), 859
handle_RustAssignment() (*angr.analyses.decompiler.structured_codegen.rust.MakeTypecast class method*), 709
handle_RustAssignment() (*angr.analyses.decompiler.structured_codegen.rust.RustStructured class method*), 708
handle_RustBinaryOp() (*angr.analyses.decompiler.structured_codegen.rust.MakeTypecast class method*), 709
handle_RustBinaryOp() (*angr.analyses.decompiler.structured_codegen.rust.PointerArithm class method*), 710
handle_RustBinaryOp() (*angr.analyses.decompiler.structured_codegen.rust.RustStructured class method*), 709
handle_RustDoWhileLoop() (*angr.analyses.decompiler.structured_codegen.rust.RustStructured class method*), 708
handle_RustForLoop() (*angr.analyses.decompiler.structured_codegen.rust.RustStructured class method*), 708
handle_RustFunction() (*angr.analyses.decompiler.structured_codegen.rust.RustStructured class method*), 708
handle_RustFunctionCall() (*angr.analyses.decompiler.structured_codegen.rust.MakeTypecast class method*), 709
handle_RustFunctionCall() (*angr.analyses.decompiler.structured_codegen.rust.RustStructured class method*), 709
handle_RustGoto() (*angr.analyses.decompiler.structured_codegen.rust.R class method*), 709
handle_RustIfBreak() (*angr.analyses.decompiler.structured_codegen.rust.RustStructured class method*), 708

handle_RustIfElse() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 708
handle_RustIndexedVariable() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 709
handle_RustInfiniteLoop() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 708
handle_RustITE() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 709
handle_RustReturn() (*angr.analyses.decompiler.structured_codegen.rust.MakeTypesExplicit class method*), 709
handle_RustReturn() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 709
handle_RustStatements() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 708
handle_RustSwitchCase() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 708
handle_RustTypeCast() (*angr.analyses.decompiler.structured_codegen.rust.FieldReferenceCollector class method*), 709
handle_RustTypeCast() (*angr.analyses.decompiler.structured_codegen.rust.MakeTypesExplicit class method*), 709
handle_RustTypeCast() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 709
handle_RustUnaryOp() (*angr.analyses.decompiler.structured_codegen.rust.MakeTypesExplicit class method*), 709
handle_RustVariableField() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 709
handle_RustVectorConvert() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 709
handle_RustVEXCCallExpression() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 709
handle_RustWhileLoop() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeWalker class method*), 708
handle_scanf() (*in module angr.analyses.reaching_definitions.function_handler_library.glibc*), 850
handle_vex_block() (*angr.engines.vex.heavy.actions.TrackActionsMixin method*), 1075
handle_vex_block() (*angr.engines.vex.heavy.inspect.SimInspectMixin method*), 1078
handle_vex_block() (*angr.engines.vex.heavy.super_fastpath.SuperFastpath method*), 1079
handle_vex_block() (*angr.engines.vex.light.light.VEXMixin method*), 1080
handle_vex_block() (*angr.engines.vex.light.slicing.VEXSlicingMixin method*), 1081
HandlerType (*class in angr.analyses.cfg.cfg_fast_msvc_eh_structs*), 345
hardcopy (*angr.state_plugins.history.TreeIter property*), 1508
has_addr_dups() (*in module angr.analyses.decompiler.utils*), 726
HAS_BATTSIGNED (*angr.analyses.code_tagging.CodeTags attribute*), 351
has_bitshifts() (*angr.analyses.code_tagging.CodeTagging method*), 751
has_blocks (*angr.rustylib.SegmentList attribute*), 1439
has_bookmark() (*angr.knowledge_plugins.bookmarks.Bookmarks method*), 1140
has_call() (*in module angr.rust.utils.ail*), 1438
has_call_in_between_stmts() (*in module angr.analyses.cfg.cfg_utils*), 1696
has_cfg (*angr.mcp.session.ProjectSession property*), 1307
has_definition() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 840
has_conflicting_value_types() (*in module angr.analyses.decompiler.ssailification.traversal_state*), 658
has_default_value (*angr.sim_state_options.StateOption property*), 1456
has_edge() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1184
has_edge_by_keys() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1184
has_fields() (*angr.rust.sim_type.EnumVariant method*), 1494
has_function_manager() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1280
has_go_hint() (*in module angr.utils.go_runtime*), 1682
has_implementation() (*angr.procedures.definitions.SimCppLibrary method*), 1329
has_implementation() (*angr.procedures.definitions.SimLibrary method*), 1328
has_implementation() (*angr.procedures.definitions.SimSyscallLibrary method*), 1332
has_index() (*in module angr.procedures.win32.local_storage*), 1385
has_invoke_target (*angr.engines.soot.statements.base.SimSootStmt property*), 1054

has_ite_expr() (*in module angr.utils.ssa*), 1696
has_job() (*angr.analyses.forward_analysis.forward_analysis method*), 764
has_jump_targets (*angr.engines.soot.statements.base.SimSootStmt property*), 1054
has_load_expr_in_between_stmts() (*in module angr.utils.ssa*), 1696
has_metadata() (*angr.procedures.definitions.SimCppLibrary method*), 1329
has_metadata() (*angr.procedures.definitions.SimLibrary method*), 1328
has_metadata() (*angr.procedures.definitions.SimSyscallLibrary method*), 1332
has_model() (*angr.knowledge_plugins.key_definitions.key_definition_manager.KeyDefinitionManager method*), 1249
has_node() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1183
has_node_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1166
has_node_addr() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1183
has_node_id() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1167
has_nonlabel_nonphi_statements() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase static method*), 720
has_nonlabel_nonphi_statements() (*in module angr.analyses.decompiler.utils*), 722
has_nonlabel_statements() (*in module angr.analyses.decompiler.utils*), 722
has_plugin() (*angr.knowledge_base.KnowledgeBase method*), 1119
has_plugin() (*angr.misc.plugins.PluginHub method*), 1318
has_plugin_preset (*angr.misc.plugins.PluginHub property*), 1318
has_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 1151
has_prototype() (*angr.procedures.definitions.SimCppLibrary method*), 1329
has_prototype() (*angr.procedures.definitions.SimLibrary method*), 1328
has_prototype() (*angr.procedures.definitions.SimSyscallLibrary method*), 1332
has_replacements_at() (*angr.knowledge_plugins.propagations.states.PropagatorState method*), 1274
has_return (*angr.knowledge_plugins.functions.function.Function property*), 1216
has_similar_stmt() (*in module angr.analyses.decompiler.block_similarity*), 398
has_some_prototype() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 1151
HAS_SQL (*angr.analyses.code_tagging.CodeTags attribute*), 351
has_sql() (*angr.analyses.code_tagging.CodeTagging method*), 352
has_statements (*angr.engines.pcode.lifter.IRSB property*), 1039
has_store() (*angr.analyses.decompiler.region_simplifiers.expr_folding.SimplifierAILEngineExpressionFolding method*), 637
has_store_stmt_in_between_stmts() (*in module angr.utils.ssa*), 1697
has_super_class() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
has_type_variable_for() (*angr.analyses.typehoon.typevars.TypeVariables method*), 918
has_unknown_bits() (*angr.sim_type.SimTypeBitfield method*), 1474
has_unresolved_calls (*angr.knowledge_plugins.functions.function.Function property*), 1213
has_unresolved_jumps (*angr.knowledge_plugins.functions.function.Function property*), 1213
has_variable() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
HAS_XOR (*angr.analyses.code_tagging.CodeTags attribute*), 351
has_xor() (*angr.analyses.code_tagging.CodeTagging method*), 351
HasCallExprWalker (*class in angr.analyses.decompiler.block_walkers*), 401
HasCallNotification, 401
HasExprWalker (*class in angr.utils.ail*), 1669
HasField (*class in angr.analyses.typehoon.typevars*), 920
HashLookupAPIDeobfuscator (*class in angr.analyses.deobfuscator.hash_lookup_api_deobfuscator*), 737
HasRefVVarNotification, 396
HasVVarNotification, 396
have_opposite_edge_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 408
header_c_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 673
heap (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
heap_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 860
heap_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1257

heap_offset (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPredicate attribute*), 1246
heap_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
HeapAddress (*class in angr.knowledge_plugins.key_definitions.heap_address*), 1248
HeapAlloc (*class in angr.procedures.win32.heap*), 1384
HeapAllocator (*class in angr.analyses.reaching_definitions.heap_allocator*), 857
HeapCreate (*class in angr.procedures.win32.heap*), 1384
HeapFree (*class in angr.procedures.win32.heap*), 1384
HeapReAlloc (*class in angr.procedures.win32.heap*), 1384
HeavyPcodeMixin (*class in angr.engines.pcode.engine*), 1036
HeavyResilienceMixin (*class in angr.engines.vex.heavy.resilience*), 1079
HeavyState (*in module angr.engines.successors*), 1064
HeavyVEXMixin (*class in angr.engines.vex.heavy.heavy*), 1078
height() (*angr.analyses.disassembly.Comment method*), 742
height() (*angr.analyses.disassembly.DisassemblyPiece method*), 740
height() (*angr.analyses.disassembly.FunctionStart method*), 740
hex_dump() (*angr.storage.memory_mixins.hex_dumper_mixin.HexDumperMixin method*), 1635
HexDumperMixin (*class in angr.storage.memory_mixins.hex_dumper_mixin*), 1635
hide_edge() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 630
hide_edge_to_successor() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 628
HIGH (*angr.analyses.language_detector.LanguageDetectionConfidenceLevel attribute*), 786
highlight() (*angr.analyses.disassembly.DisassemblyPiece method*), 740
history_contains() (*angr.state_hierarchy.StateHierarchy method*), 1494
history_predecessors() (*angr.state_hierarchy.StateHierarchy method*), 1494
history_successors() (*angr.state_hierarchy.StateHierarchy method*), 1494
HistoryIter (*class in angr.state_plugins.history*), 1563
HistoryTrackingMixin (*class in angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin*), 1648
HITMAP_SIZE (*angr.state_plugins.edge_hitmap.SimStateEdgeHitmap attribute*), 1547
HITMAP_SIZE (*angr.state_plugins.SimStateEdgeHitmap attribute*), 1526
hooks() (*angr.knowledge_plugins.functions.function.Function method*), 1218
Hook (*class in angr.analyses.disassembly*), 740
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.PutHook static method*), 336
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegisterInitia method*), 336
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.ShrBoundsC static method*), 336
hook() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.StoreHook static method*), 336
hook() (*angr.analyses.reaching_definitions.function_handler.FunctionHan method*), 843
hook() (*angr.project.Project method*), 1390
hook() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
hook_add() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 1603
hook_after() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.Load method*), 336
hook_before() (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.Loa method*), 336
hook_del() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 1603
hook_reset() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 1603
hook_symbol() (*angr.project.Project method*), 1391
hooked_by() (*angr.project.Project method*), 1390
HookedMethod (*class in angr.misc.hookset*), 1316
HookNode (*class in angr.codenode*), 988
HookSet (*class in angr.misc.hookset*), 1315
HooksMixin (*class in angr.engines.hook*), 1005
htonl (*class in angr.procedures.posix.htonl*), 1372
htons (*class in angr.procedures.posix.htons*), 1372

## I

IcicleEngine (*class in angr.engines.icicle*), 1005
IcicleStateTranslationData (*class in angr.state_plugins.icicle*), 1563
IcicleVMRef (*class in angr.state_plugins.icicle*), 1564
id (*angr.engines.soot.values.thisref.SimSootValue_ThisRef property*), 1062
ident (*angr.analyses.disassembly.DisassemblyPiece attribute*), 739
ident (*angr.analyses.disassembly.OperandPiece attribute*), 742
identical_blocks (*angr.analyses.bindiff.BinDiff property*), 282

identical_blocks (*angr.analyses.bindiff.FunctionDiff property*), 281
identical_functions (*angr.analyses.bindiff.BinDiff property*), 282
Identifier (*class in angr.analyses.identifier.identify*), 784
IdentifierCallable (*class in angr.analyses.identifier.custom_callable*), 770
IdentifierException, 770
identify_func() (*angr.analyses.identifier.identify.Identifier method*), 785
idom() (*angr.utils.doms.IncrementalDominators method*), 1677
idx (*angr.ailment.Block attribute*), 210
idx (*angr.ailment.block.Block attribute*), 215
IfElseFlattener (*class in angr.analyses.decompiler.region_simplifiers.ifelse*), 638
IfLetNode (*class in angr.rust.structuring.structurer_nodes*), 1436
IfSimplifier (*class in angr.analyses.decompiler.region_simplifiers.if_*), 638
IFUNC_HINTS (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 319
IFuncResolver (*class in angr.procedures.linux_loader.sim_loader*), 1368
immediate_dominators() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 313
immediate_postdominators() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 313
ImportedLine (*class in angr.analyses.decompiler.structured_codegen.dwarf_import*), 691
ImportSourceCode (*class in angr.analyses.decompiler.structured_codegen.dwarf_import*), 691
IN (*angr.sim_type.PointerDisposition attribute*), 1466
in_degree (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1185
in_edges (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1185
in_other() (*in module angr.analyses.decompiler.block_similarity*), 399
IN_OUT (*angr.sim_type.PointerDisposition attribute*), 1466
IN_OUTMAYBE (*angr.sim_type.PointerDisposition attribute*), 1466
IN_SCOPE (*angr.misc.Loggers attribute*), 1310
IN_SCOPE (*angr.misc.loggers.Loggers attribute*), 1316
inc_active_workers() (*angr.distributed.server.Server method*), 994
includes() (*angr.keyed_region.RegionObject method*), 1116
includes() (*angr.storage.memory_object.SimMemoryObject method*), 1667
includes_function() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 834
IncompleteSwitchCaseHeadStatement (*class in angr.analyses.decompiler.structurer_nodes*), 713
IncompleteSwitchCaseNode (*class in angr.analyses.decompiler.structurer_nodes*), 713
inconsistent (*angr.analyses.stack_pointer_tracker.StackPointerTracker property*), 896
inconsistent_for() (*angr.analyses.stack_pointer_tracker.StackPointerT method*), 896
IncorrectLocationException, 1047
increase_lmdb_map_size() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method*), 1278
IncrementalDominators (*class in angr.utils.doms*), 1677
indent_str() (*angr.analyses.decompiler.structured_codegen.c.CConstruc static method*), 672
indent_str() (*angr.analyses.decompiler.structured_codegen.rust.RustCo static method*), 692
index() (*angr.utils.ins_addr_list.InsAddrList method*), 1690
index_key_func_addrs() (*angr.knowledge_plugins.functions.function_manager.FunctionM method*), 1227
index_of_similar_stmts() (*in module angr.analyses.decompiler.block_similarity*), 399
IndirectJump (*class in angr.knowledge_plugins.cfg.indirect_jump*), 1176
IndirectJump (*class in angr.protos.cfg_pb2*), 1395
IndirectJumpResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.resolver*), 340
IndirectJumps (*class in angr.knowledge_plugins.indirect_jumps*), 1228
IndirectJumpType (*class in angr.knowledge_plugins.cfg.indirect_jump*), 1175
INET_INADDRSTRLEN (*angr.procedures.posix.inet_ntoa.inet_ntoa attribute*), 1372
inet_ntoa (*class in angr.procedures.posix.inet_ntoa*),

1372
infer_provider() (*angr.llm_client.LLMClient method*), 1293
infer_shapes() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 903
INFERRED (*angr.knowledge_plugins.callsite_prototypes.CallSitePrototypes attribute*), 1150
INFO (*angr.analyses.decompiler.notes.decompilation_note.DecompilationNoteLevel attribute*), 538
INFO (*angr.analyses.decompiler.notes.DecompilationNoteLevel attribute*), 536
info (*angr.knowledge_plugins.functions.function.Function property*), 1211
init_class() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 1573
init_dfs() (*angr.utils.doms.IncrementalDominators method*), 1677
init_doms() (*angr.utils.doms.IncrementalDominators method*), 1677
init_hierarchy() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
init_replacements() (*angr.knowledge_plugins.propagations.states.PropagatorState method*), 1274
init_state() (*angr.state_plugins.plugin.SimStatePlugin method*), 1579
init_static_field() (*angr.simos.javavm.SimJavaVM static method*), 1487
init_str() (*angr.procedures.definitions.SimTypeCollection method*), 1324
INITIAL_SP_16BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 812
INITIAL_SP_16BIT (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 1250
INITIAL_SP_16BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 1229
INITIAL_SP_32BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 812
INITIAL_SP_32BIT (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 1250
INITIAL_SP_32BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 1230
INITIAL_SP_64BIT (*angr.analyses.reaching_definitions.LiveDefinitions attribute*), 812
INITIAL_SP_64BIT (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions attribute*), 1250
INITIAL_SP_64BIT (*angr.knowledge_plugins.key_definitions.LiveDefinitions attribute*), 1230
initial_state() (*angr.analyses.purity.engine.PurityEngineAIL method*), 803
initial_state() (*angr.knowledge_plugins.propagations.states.PropagatorState class method*), 1273
initial_state() (*angr.knowledge_plugins.propagations.states.PropagatorVEXState class method*), 1275
INITIALIZATION (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
InitializationFinder (*class in angr.analyses.init_finder*), 786
initialize() (*angr.utils.mp.Initializer method*), 1694
initialize_killed_function_arguments() (*angr.analyses.reaching_definitions.rd_initializer.RDAStateInitial method*), 858
initialize_architectural_state() (*angr.analyses.reaching_definitions.rd_initializer.RDAStateInitial method*), 859
initialize_dominance_frontiers() (*angr.analyses.variable_recovery.variable_recovery_base.Variabl method*), 927
initialize_function_state() (*angr.analyses.reaching_definitions.rd_initializer.RDAStateInitial method*), 857
initialize_one_function_argument() (*angr.analyses.reaching_definitions.rd_initializer.RDAStateInitial method*), 858
initialize_stack_pointer() (*angr.analyses.reaching_definitions.rd_initializer.RDAStateInitial method*), 859
initialize_variable_names() (*angr.knowledge_plugins.variables.variable_manager.VariableMa method*), 1289
InitializeCriticalSectionAndSpinCount (*class in angr.procedures.win32.critical_section*), 1383
InitializeCriticalSectionEx (*class in angr.procedures.win32.critical_section*), 1383
initialized_classes (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader property*), 1573
Initializer (*class in angr.utils.mp*), 1694
InitialValueTag (*class in angr.knowledge_plugins.key_definitions.tag*), 1261
inline_call() (*angr.procedures.SimProcedure method*), 1451
inline_comment() (*angr.knowledge_plugins.comments.Comments method*), 1190
InlinedMemcpySimplifier (*class in angr.analyses.decompiler.optimization_passes.inlined_memcpy_s 577*
InlinedMemcpySimplifierLate (*class in angr.analyses.decompiler.optimization_passes.inlined_memcpy_s 577*
InlinedMemsetSimplifier (*class in angr.analyses.decompiler.optimization_passes.inlined_memset_si 578*
InlinedMemsetSimplifierLate (*class in angr.analyses.decompiler.optimization_passes.inlined_memset_si 578*

InlinedStrcpySimplifier (*class in angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier*), 578
InlinedStrcpySimplifierLate (*class in angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier*), 579
InlinedStringTransformationAILEngine (*class in angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier*), 580
InlinedStringTransformationSimplifier (*class in angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier*), 581
InlinedStringTransformationState (*class in angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier*), 579
InlinedStrlenSimplifier (*class in angr.analyses.decompiler.optimization_passes.inlined_strlen_simplifier*), 581
InlinedWcscpySimplifier (*class in angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier*), 581
InlinedWcscpySimplifierLate (*class in angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier*), 582
InlineStringTransformationDescriptor (*class in angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier*), 580
inner_step() (*angr.exploration_techniques.threading.Threading method*), 1107
input_variables() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1287
ins_addr (*angr.ailment.Manager attribute*), 212
ins_addr (*angr.ailment.manager.Manager attribute*), 224
ins_addr (*angr.code_location.AILCodeLocation property*), 986
ins_addr (*angr.errors.SimError attribute*), 1083
ins_addr (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 1245
ins_addr (*angr.SimError attribute*), 177
ins_addrs (*angr.state_plugins.history.SimStateHistory property*), 1562
ins_sizes (*angr.utils.ins_addr_list.InsAddrList property*), 1690
InsAddrList (*class in angr.utils.ins_addr_list*), 1689
InsAddrSeq (*class in angr.utils.vex*), 1703
Insert (*class in angr.ailment.expression*), 220
insert() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
insert() (*angr.state_plugins.filesystem.SimMount method*), 1550
insert_asm() (*angr.analyses.reassembler.Reassembler method*), 876
insert_node() (*angr.analyses.decompiler.structurer_nodes.SequenceNode method*), 711
insert_node() (*angr.analyses.decompiler.utils.simplify*), 721
insert_phi_statements() (*angr.analyses.decompiler.ssailification.rewriting.RewritingAnaly static method*), 653
inserted_asm_after_label (*angr.analyses.reassembler.Reassembler property*), 875
inserted_asm_before_label (*angr.analyses.reassembler.Reassembler property*), 875
InsertLowBitsReadRewriter (*class in angr.analyses.decompiler.simplifier*), 397
insn_addr (*angr.code_location.AILCodeLocation property*), 986
insn_addr_to_memory_data (*angr.analyses.cfg.cfg_fast.CFGFast property*), 225
insn_addr_unwrap (*angr.code_location.AILCodeLocation property*), 986
install_observer() (*angr.analyses.reaching_definitions.reaching_definitions. method*), 868
InspectAttrs (*class in angr.state_plugins.inspect*), 1566
InspectMixin (*class in angr.storage.memory_mixins.inspect_mixin*), 1636
install_hooks() (*angr.SimManagerInternalHookSet static method*), 1315
instantiate() (*angr.analyses.decompiler.known_patterns.templates.Kno method*), 532
INSTRUCTION (*angr.state_plugins.inspect.EventType attribute*), 1565
instruction (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
instruction (*angr.state_plugins.InspectAttrs attribute*), 1498
Instruction (*class in angr.analyses.disassembly*), 740
Instruction (*class in angr.analyses.reassembler*), 872
Instruction (*class in angr.protos.primitives_pb2*), 1404
instruction_addresses (*angr.engines.pcode.lifter.IRSB property*), 1040
instruction_addresses() (*angr.analyses.reassembler.BasicBlock method*), 873
instruction_addresses() (*angr.analyses.reassembler.Procedure method*), 874
instruction_addrs (*angr.block.Block property*), 961
INSTRUCTION_LIMIT (*angr.emulator.EmulatorStopReason attribute*), 996

INSTRUCTION_LIMIT (*angr.EmulatorStopReason attribute*), 161
instruction_size() (*angr.knowledge_plugins.functions.function.Function method*), 1216
InstructionError, 870
InstructionMapping (*class in angr.analyses.decompiler.structured_codegen.base*), 669
InstructionMappingElement (*class in angr.analyses.decompiler.structured_codegen.base*), 669
InstructionMappingEntry (*class in angr.protos.codegen_pb2*), 1401
InstructionMappingMsg (*class in angr.protos.codegen_pb2*), 1401
instructions (*angr.analyses.reassembler.Reassembler property*), 875
instructions (*angr.block.Block property*), 961
instructions (*angr.engines.pcode.lifter.IRSB property*), 1040
instrument() (*angr.analyses.smc.TraceClassifier method*), 892
instrument() (*angr.procedures.posix.readdir.readdir method*), 1374
instrument() (*angr.procedures.win32.sim_time.GetLocalTime method*), 1386
instrument() (*angr.procedures.win32.sim_time.GetSystemTimeAsFileTime method*), 1386
INT (*angr.analyses.decompiler.structured_codegen.base.CConstantType attribute*), 668
Int (*class in angr.analyses.typehoon.typeconsts*), 907
Int1 (*class in angr.analyses.typehoon.typeconsts*), 907
Int128 (*class in angr.analyses.typehoon.typeconsts*), 909
Int16 (*class in angr.analyses.typehoon.typeconsts*), 907
Int256 (*class in angr.analyses.typehoon.typeconsts*), 909
int2base() (*in module angr.analyses.identifier.functions.based_atoi*), 772
int2base() (*in module angr.analyses.identifier.functions.strtol*), 784
int2base() (*in module angr.state_plugins.trace_additions*), 1597
int2str (*class in angr.analyses.identifier.functions.int2str*), 773
int2str_v2 (*class in angr.analyses.identifier.functions.int2str*), 774
int2str_v3 (*class in angr.analyses.identifier.functions.int2str*), 774
int2str_v4 (*class in angr.analyses.identifier.functions.int2str*), 774
Int32 (*class in angr.analyses.typehoon.typeconsts*), 908
Int512 (*class in angr.analyses.typehoon.typeconsts*), 909
Int64 (*class in angr.analyses.typehoon.typeconsts*), 908
Int8 (*class in angr.analyses.typehoon.typeconsts*), 907
int_args (*angr.calling_conventions.SimCC property*), 1068
int_len_mod (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1379
int_sign (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1379
int_type() (*in module angr.analyses.typehoon.typeconsts*), 913
Integer (*angr.analyses.proximity_graph.ProximityNodeTypes attribute*), 799
Integer (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
Integer (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
Integer (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
IntegerProximityNode (*class in angr.analyses.proximity_graph*), 800
IntegerToInt (*class in angr.procedures.java_lang.integer*), 1348
IntegerToString (*class in angr.procedures.java_lang.integer*), 1348
IntegerValueOf (*class in angr.procedures.java_lang.integer*), 1348
interference_graph() (*angr.analyses.s_liveness.SLivenessAnalysis method*), 878
InterferenceChecker (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 636
InterlockedExchange (*class in angr.procedures.win32.InterlockedExchange*), 1383
intern_simvar() (*angr.analyses.decompiler.structured_codegen.c_serial method*), 688
intern_tags() (*angr.analyses.decompiler.structured_codegen.c_serialize method*), 688
intern_type() (*angr.analyses.decompiler.structured_codegen.c_serialize method*), 688
internal_objects (*angr.keyed_region.RegionObject property*), 1116
interpret() (*angr.procedures.stubs.format_parser.FormatString method*), 1378
intersection() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1587
intrinsic_name() (*angr.analyses.decompiler.structured_codegen.c.CDir method*), 684
IntToInteger (*class in angr.procedures.java_lang.integer*), 1348
IntVar (*class in angr.analyses.typehoon.typeconsts*), 909
InvalICache (*angr.protos.primitives_pb2.Edge attribute*), 1403
InvalidArgumentError, 1301

invalidate() (*angr.angrdb.db.AngrDB method*), 941
invalidate() (*angr.knowledge_plugins.comments.Comments method*), 1191
invalidate() (*in module angr.analyses.decompiler.edits.cache*), 436
invalidate_direct_next() (*angr.engines.pcode.lifter.IRSB method*), 1040
InvalidNameError, 437
inverse() (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode method*), 902
inverse_variant() (*angr.rust.optimization_passes.pre_pattern_match_simplifier.PrePatternMatchSimplifier static method*), 1427
inverse_wo_tag() (*angr.analyses.typehoon.simple_solver.ConstraintGraphNode method*), 902
inverted() (*angr.analyses.typehoon.simple_solver.TypeLattice method*), 899
inverted_idoms() (*in module angr.utils.graph*), 1683
InvertNegatedLogicalConjunctionsAndDisjunctions (*class in angr.analyses.decompiler.peephole_optimizations.invert_negated_logical_conjuction_disjunction*), 612
InvokeBase (*class in angr.engines.soot.expressions.invoke*), 1051
ip (*angr.sim_state.SimState property*), 1454
irange() (*angr.knowledge_plugins.functions.function_manager.FunctionDict method*), 1219
irange() (*angr.knowledge_plugins.variables.variable_manager.VariableDict method*), 1283
irange() (*angr.utils.tagged_interval_map.TaggedIntervalMap method*), 1700
IROp (*class in angr.analyses.disassembly*), 740
IRSB (*angr.state_plugins.inspect.EventType attribute*), 1565
IRSB (*class in angr.engines.pcode.lifter*), 1037
irsb_from_node() (*angr.analyses.vfg.VFG method*), 936
IRSBConverter (*class in angr.ailment*), 212
IRSBRegisterCollector (*class in angr.analyses.deobfuscator.irsb_reg_collector*), 737
IRTop (*class in angr.engines.light.engine*), 1010
is_a_jump_target() (*angr.analyses.decompiler.structuring.structurer_nodes.MultiNode static method*), 720
is_addr_known (*angr.codenode.FuncNode property*), 988
is_ail (*angr.state_plugins.scratch.SimStateScratch property*), 1583
is_alignment (*angr.knowledge_plugins.functions.function.Function property*), 1211
is_alignment_mask() (*in module angr.utils.constants*), 1675
is_anonymous_struct() (*in module angr.procedures.definitions.parse_win32json*), 1334
is_apiname() (*angr.analyses.deobfuscator.api_obf_finder.APIObfuscationFinder static method*), 735
is_bit_set_be() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed static method*), 760
is_bool_expr() (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExprBase static method*), 605
is_bounded() (*angr.state_plugins.uc_manager.SimUCManager method*), 1598
is_cached() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method*), 1180
is_cached() (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict method*), 1220
is_call_or_call_assignment() (*angr.analyses.deobfuscator.string_obf_opt_passes.StringObfType2Transformer static method*), 739
is_class_initialized() (*angr.state_plugins.javavm_classloader.SimJavaVmClassloader method*), 1573
is_composite_type() (*in module angr.rust.sim_type*), 1431
is_concrete() (*angr.analyses.init_finder.SimEngineInitFinderVEX static method*), 786
is_const_and_vvar_assignment() (*in module angr.utils.ssa*), 1697
is_const_assignment() (*in module angr.utils.ssa*), 1697
is_const_dirty_load_assignment() (*in module angr.utils.ssa*), 1697
is_const_vvar_load_dirty_assignment() (*in module angr.utils.ssa*), 1697
is_cpp_funcname_ctor() (*in module angr.utils.cpp*), 1677
is_cross_referenced() (*angr.analyses.vtable.VtableFinder method*), 937
is_dump (*angr.simos.windows.SimWindows property*), 1492
is_empty (*angr.engines.successors.SimSuccessors property*), 1063
is_empty (*angr.keyed_region.RegionObject property*), 1116
is_empty (*angr.storage.memory_mixins.regioned_memory.region_data.RegionBase property*), 1660
is_empty() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink method*), 349
is_empty_node() (*in module angr.analyses.decompiler.utils*), 722
is_empty_or_label_only_node() (*in module angr.analyses.decompiler.utils*), 722
is_enabled_for() (*in module angr.misc.loggers*), 1317
is_expr_used_as_reg_base_value() (*in module angr.utils.ail*), 1670
is_extern (*angr.code_location.AILCodeLocation property*), 986

is_false() (*angr.state_plugins.solver.SimSolver method*), 1591
is_fp_arg() (*angr.calling_conventions.SimCC method*), 968
is_fp_value() (*angr.calling_conventions.SimCC static method*), 969
is_free() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
is_func_nonreturning() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1224
is_func_returning_unknown() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1224
IS_FUNCTION (*angr.procedures.ntdll.exceptions.KiUserExceptionDispatcher attribute*), 1369
IS_FUNCTION (*angr.sim_procedure.SimProcedure attribute*), 1450
IS_FUNCTION (*angr.SimProcedure attribute*), 190
is_function() (*angr.analyses.vtable.VtableFinder method*), 937
is_function_argument (*angr.sim_variable.SimVariable property*), 1478
is_function_entry() (*angr.knowledge_plugins.comments.Comments method*), 1190
is_function_likely_security_init_cookie() (*in module angr.utils.funcid*), 1680
is_function_security_check_cookie() (*in module angr.utils.funcid*), 1680
is_function_security_check_cookie_strict() (*in module angr.utils.funcid*), 1680
is_function_security_init_cookie() (*in module angr.utils.funcid*), 1680
is_function_security_init_cookie_win8() (*in module angr.utils.funcid*), 1680
is_global_variable_address() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 928
is_global_variable_updated() (*angr.analyses.s_propagator.SPropagator static method*), 879
is_go_binary (*angr.project.Project property*), 1392
is_go_noreturn_name() (*in module angr.utils.go_runtime*), 1682
is_goto_edge() (*angr.analyses.decompiler.goto_manager.GotoManager method*), 452
is_head_controlled_loop_block() (*in module angr.utils.ail*), 1670
is_heap_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState static method*), 861
is_heap_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1257
is_hex() (*in module angr.analyses.reassembler*), 870
is_hook (*angr.codenode.BlockNode attribute*), 987
is_hook (*angr.codenode.CodeNode attribute*), 987
is_hook (*angr.codenode.HookNode attribute*), 988
is_hook (*angr.codenode.SyscallNode attribute*), 988
is_hooked() (*angr.project.Project method*), 1390
is_immediate (*angr.analyses.reassembler.Operand property*), 872
is_in_readonly_section() (*in module angr.utils.loader*), 1693
is_in_readonly_segment() (*in module angr.utils.loader*), 1693
is_inlined_strcpy() (*angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier.InlinedStrcpy method*), 579
is_inlined_wcsncpy() (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier.InlinedWcscpy method*), 582
is_integer_likely_a_string() (*angr.analyses.decompiler.optimization_passes.inlined_strcpy_simplifier.InlinedStrcpy static method*), 579
is_integer_likely_a_wide_string() (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier.InlinedWcscpy static method*), 582
is_java (*angr.procedures.java.JavaSimProcedure property*), 1337
is_java (*angr.sim_procedure.SimProcedure property*), 1452
is_leaf() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1586
is_libname() (*angr.analyses.deobfuscator.api_obf_finder.APIObfuscationFinder static method*), 735
is_linux_kernel_object (*angr.analyses.decompiler.known_patterns.context.PatternContext attribute*), 488
is_linux_kernel_object (*angr.analyses.decompiler.known_patterns.PatternContext attribute*), 478
is_live (*angr.sim_state_pickle.SimStatePickle property*), 1565
is_lsb_extract() (*in module angr.ailment.utils*), 227
is_lsb_overwrite() (*in module angr.ailment.utils*), 228
is_machine_word_size_type() (*in module angr.analyses.decompiler.structured_codegen.c*), 671
is_machine_word_size_type() (*in module angr.analyses.decompiler.structured_codegen.rust*), 692
is_none_or_likeable() (*in module angr.utils.ail*), 227
is_none_or_matchable() (*in module angr.utils.ail*), 227
is_noop_block (*angr.engines.pcode.lifter.IRSB property*),

*erty*), 1041
is_not_in_cfg() (*angr.analyses.veritesting.Veritesting method*), 931
is_occupied() (*angr.rustylib.SegmentList method*), 1439
is_on_stack (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation property*), 1245
is_overbound() (*angr.analyses.veritesting.Veritesting method*), 932
is_pc() (*in module angr.utils.loader*), 1693
is_phi_assignment() (*in module angr.utils.ail*), 1669
is_phi_assignment() (*in module angr.utils.ssa*), 1697
is_phi_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1286
is_phi_vvar_id() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 886
is_plt (*angr.analyses.reassembler.Procedure property*), 873
is_plt (*angr.knowledge_plugins.functions.function.Function property*), 1211
is_plt_cached() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1221
is_prev_free() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 1559
is_prototype_certain() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 1151
is_prototype_groundtruth (*angr.knowledge_plugins.functions.function.Function property*), 1211
is_prototype_guessed (*angr.knowledge_plugins.functions.function.Function property*), 1211
is_prototype_manual() (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypes method*), 1151
is_pure() (*angr.analyses.purity.engine.ResultType method*), 802
is_pyinstaller() (*in module angr.utils.env*), 1679
is_returning() (*angr.procedures.definitions.SimLibrary method*), 1328
is_rust_binary (*angr.project.Project property*), 1392
is_rust_function() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1217
is_sane_register_variable() (*in module angr.analyses.calling_convention.utils*), 289
is_scalar_int_convert() (*in module angr.utils.ail*), 1669
is_signed (*angr.engines.vex.claripy.irop.SimIROp property*), 1074
is_similar() (*in module angr.analyses.decompiler.block_similarity*), 398
is_simple_jump_node() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl* 640
is_simprocedure (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
is_simprocedure (*angr.knowledge_plugins.functions.function.Function property*), 1211
is_stack (*angr.storage.memory_mixins.regioned_memory.region_meta_m property*), 1662
is_stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachiDefinitionsState method*), 861
is_stack_address() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase static method*), 928
is_stack_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1251
is_stack_probe() (*in module angr.calling_conventions*), 984
is_stackarg_vvar() (*angr.analyses.decompiler.stackarg_offset_manager.StackargOffsetManager method*), 660
is_statement_terminating() (*in module angr.analyses.decompiler.utils*), 724
is_stored() (*angr.sim_manager.SimulationManager method*), 1704
is_subclass() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
is_subclass_including() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
is_symbol_hooked() (*angr.project.Project method*), 1391
is_symbolic (*angr.state_plugins.sim_action.SimAction property*), 1584
is_symbolic (*angr.state_plugins.sim_action.SimActionConstraint property*), 1585
is_symbolic (*angr.state_plugins.sim_action.SimActionData property*), 1585
is_symbolic (*angr.state_plugins.sim_action.SimActionExit property*), 1585
is_symbolic (*angr.state_plugins.sim_action.SimActionOperation property*), 1585
is_syscall (*angr.analyses.cfg.cfg_emulated.CFGJob property*), 310
is_syscall (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
is_syscall (*angr.knowledge_plugins.functions.function.Function property*), 1211
is_syscall_addr() (*angr.simos.simos.SimOS method*), 1490
is_syscall_addr() (*angr.simos.userland.SimUserland method*), 1491
is_syscall_addr() (*angr.simos.windows.SimWindows method*), 1493
is_taint_impacting_stack_pointers() (*angr.analyses.backward_slice.BackwardSlice method*), 278
is_taint_related_to_ip()

(*angr.analyses.backward_slice.BackwardSlice method*), 278
is_thumb_addr() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
is_top() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 860
is_top() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase static method*), 928
is_top() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1251
is_top() (*angr.knowledge_plugins.propagations.states.PropagationState static method*), 1273
is_true() (*angr.state_plugins.solver.SimSolver method*), 1590
is_va_start_amd64() (*angr.analyses.calling_convention.calling_convention.CallingConvention method*), 286
is_variable_used_at() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1284
is_visible_class() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
is_visible_method() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
is_vvar_eliminatable() (*in module angr.utils.ssa*), 1698
is_vvar_used_for_addr_loading_switch_case() (*angr.analyses.s_propagator.SPropagator static method*), 879
is_windows_kernel_driver (*angr.analyses.decompiler.known_patterns.contextPatternContext attribute*), 488
is_windows_kernel_driver (*angr.analyses.decompiler.known_patterns.PatternContext attribute*), 478
isalnum (*class in angr.procedures.libc.isalnum*), 1356
isalpha (*class in angr.procedures.libc.isalpha*), 1356
IsArray (*class in angr.analyses.typehoon.typevars*), 920
isascii (*class in angr.procedures.libc.isascii*), 1357
IsBadCodePtr (*class in angr.procedures.win32.is_bad_ptr*), 1384
IsBadReadPtr (*class in angr.procedures.win32.is_bad_ptr*), 1384
IsBadWritePtr (*class in angr.procedures.win32.is_bad_ptr*), 1384
isblank (*class in angr.procedures.libc.isblank*), 1357
iscntrl (*class in angr.procedures.libc.iscntrl*), 1357
isdigit (*class in angr.procedures.libc.isdigit*), 1357
isgraph (*class in angr.procedures.libc.isgraph*), 1357
IsInstanceOf (*class in angr.procedures.java_jni.object_operations*), 1347
islower (*class in angr.procedures.libc.islower*), 1357
ISPOMixin (*class in angr.storage.memory_mixins.paged_memory.pages.isp* 1648
isprint (*class in angr.procedures.libc.isprint*), 1357
IsProcessorFeaturePresent (*class in angr.procedures.win32.IsProcessorFeaturePresent*), 1383
ispunct (*class in angr.procedures.libc.ispunct*), 1357
IsSameObject (*class in angr.procedures.java_jni.object_operations*), 1347
isspace (*class in angr.procedures.libc.isspace*), 1357
isupper (*class in angr.procedures.libc.isupper*), 1357
isxdigit (*class in angr.procedures.libc.isxdigit*), 1357
ITE (*class in angr.ailment.expression*), 218
IteExpr (*class in angr.protos.ail_types_pb2*), 1394
ITEExprConverterAnalysis (*class in angr.analyses.decompiler.optimization_passes.ite_expr_converter* 582
IteExprs (*class in angr.protos.ail_types_pb2*), 1394
items() (*angr.analyses.ddg.LiveDefinitions method*), 363
items() (*angr.analyses.decompiler.structured_codegen.base.InstructionMa method*), 669
items() (*angr.analyses.decompiler.structured_codegen.base.PositionMapp method*), 668
items() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method*), 1180
items() (*angr.knowledge_plugins.functions.function_manager.FunctionDic method*), 1218
items() (*angr.knowledge_plugins.labels.Labels method*), 1264
items() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
items() (*angr.state_plugins.globals.SimStateGlobals method*), 1551
items() (*angr.storage.memory_mixins.paged_memory.pages.multi_values. method*), 1650
iter_comments() (*angr.knowledge_plugins.comments.Comments method*), 1191
iter_own() (*angr.knowledge_plugins.types.TypesStore method*), 1280
iter_own_keys() (*angr.knowledge_plugins.types.TypesStore method*), 1280
IteratorHasNext (*class in angr.procedures.java_util.iterator*), 1350
IteratorNext (*class in angr.procedures.java_util.iterator*), 1350
ITERegionConverter (*class in angr.analyses.decompiler.optimization_passes.ite_region_convert* 583
itervariables() (*angr.analyses.ddg.LiveDefinitions method*), 363

## J


JavaSimProcedure (*class in angr.procedures.java*), 1337
javavm_memory (*angr.sim_state.SimState property*), 1454
javavm_registers (*angr.sim_state.SimState property*), 1454
JavaVmMemory (*class in angr.storage.memory_mixins*), 1621
JavaVmMemoryMixin (*class in angr.storage.memory_mixins.javavm_memory_mixin*), 1636
JNI_ABORT (*angr.procedures.java_jni.array_operations.ReleaseArrayElements attribute*), 1339
JNI_COMMIT (*angr.procedures.java_jni.array_operations.ReleaseArrayElements attribute*), 1339
JNI_FALSE (*angr.procedures.java_jni.JNISimProcedure attribute*), 1338
JNI_TRUE (*angr.procedures.java_jni.JNISimProcedure attribute*), 1338
JNISimProcedure (*class in angr.procedures.java_jni*), 1338
job (*angr.analyses.forward_analysis.job_info.JobInfo property*), 764
JobInfo (*class in angr.analyses.forward_analysis.job_info*), 764
jobs (*angr.analyses.forward_analysis.forward_analysis.ForwardAnalysis property*), 764
join() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 903
join_simtypes() (*angr.analyses.typehoon.simple_solver.SimpleSolver class method*), 904
json_decode() (*in module angr.utils.json_utils*), 1690
json_encode() (*in module angr.utils.json_utils*), 1690
json_serialize_load_args() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 954
Jump (*class in angr.ailment.statement*), 225
jump() (*angr.sim_procedure.SimProcedure method*), 1452
jump_guards (*angr.state_plugins.history.SimStateHistory property*), 1562
jump_sources (*angr.state_plugins.history.SimStateHistory property*), 1562
jump_tables (*angr.analyses.cfg.cfg_fast.CFGFast property*), 325
jump_targets (*angr.state_plugins.history.SimStateHistory property*), 1562
jumpkinds (*angr.state_plugins.history.SimStateHistory property*), 1562
jumpout_sites (*angr.knowledge_plugins.functions.function.Function property*), 1213
jumptable_addr (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 1177
Jumptable_AddressComputed (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 1175
Jumptable_AddressComputed (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 1163
Jumptable_AddressLoadedFromMemory (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 1175
Jumptable_AddressLoadedFromMemory (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 1163
jumptable_entries (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJ property*), 1177
jumptable_entries_guessed (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 1177
jumptable_entry_size (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 1177
jumptable_size (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJump property*), 1177
JumpTableEntryConditionRewriter (*class in angr.analyses.decompiler.jumptable_entry_condition_rewriter*), 453
JumptableInfo (*class in angr.knowledge_plugins.cfg.indirect_jump*), 1175
JumptableInfo (*class in angr.protos.cfg_pb2*), 1395
JumpTableProcessor (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 335
JumpTableProcessorState (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 335
JumpTableResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 337
JumpTargetBaseAddr (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 334
JumpTargetCollector (*class in angr.analyses.decompiler.jump_target_collector*), 453


## K


kb (*angr.analyses.decompiler.optimization_passes.optimization_pass.BaseO property*), 587
kb (*angr.project.Project property*), 1389
keep_path() (*angr.annocfg.AnnotatedCFG method*), 958
KEY (*angr.procedures.libc.strtok.strtok attribute*), 1361
KEY (*angr.procedures.linux_kernel.time.time attribute*), 1367

KEY (*angr.procedures.win32.dynamic_loading.GetProcAddress attribute*), 1384
KEY (*angr.procedures.win32.local_storage.FlsAlloc attribute*), 1385
KEY (*angr.procedures.win32.local_storage.FlsFree attribute*), 1385
KEY (*angr.procedures.win32.local_storage.FlsGetValue attribute*), 1385
KEY (*angr.procedures.win32.local_storage.FlsSetValue attribute*), 1385
KEY (*angr.procedures.win32.local_storage.TlsAlloc attribute*), 1385
KEY (*angr.procedures.win32.local_storage.TlsFree attribute*), 1385
KEY (*angr.procedures.win32.local_storage.TlsGetValue attribute*), 1385
KEY (*angr.procedures.win32.local_storage.TlsSetValue attribute*), 1385
KEY (*angr.procedures.win32.sim_time.GetTickCount64 attribute*), 1386
key (*angr.sim_variable.SimConstantVariable property*), 1478
key (*angr.sim_variable.SimMemoryVariable property*), 1479
key (*angr.sim_variable.SimRegisterVariable property*), 1478
key (*angr.sim_variable.SimStackVariable property*), 1479
key (*angr.sim_variable.SimTemporaryVariable property*), 1478
key (*angr.sim_variable.SimVariable property*), 1478
KeyDefinitionManager (*class in angr.knowledge_plugins.key_definitions.key_definition_manager*), 1249
KeyedRegion (*class in angr.keyed_region*), 1116
keys() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict method*), 1180
keys() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
keys() (*angr.state_plugins.globals.SimStateGlobals method*), 1551
keys() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 1650
keys() (*angr.vaults.Vault method*), 1704
KeyValueMemory (*class in angr.storage.memory_mixins*), 1622
KeyValueMemoryMixin (*class in angr.storage.memory_mixins.keyvalue_memory_mixin*), 1638
kill_and_add_definition() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 863
kill_and_add_definition() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1253
kill_def() (*angr.analyses.ddg.LiveDefinitions method*), 363
kill_def() (*angr.knowledge_plugins.key_definitions.liveness.Liveness method*), 1257
kill_def() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingD method*), 1259
kill_definitions() (*angr.analyses.reaching_definitions.rd_state.Reachi method*), 863
kill_definitions() (*angr.knowledge_plugins.key_definitions.live_defini method*), 1253
kind (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseHe attribute*), 713
kind (*angr.knowledge_plugins.key_definitions.definition.DefinitionMatchPr attribute*), 1245
Kind (*in module angr.analyses.decompiler.ssailification.ssailification*), 655
kind_of() (*angr.knowledge_plugins.comments.Comments method*), 1190
kinds (*angr.knowledge_plugins.comments.Comments property*), 1190
KiUserExceptionDispatcher (*class in angr.procedures.ntdll.exceptions*), 1369
KnowledgeBase (*class in angr.knowledge_base*), 1118
KnowledgeBasePlugin (*class in angr.knowledge_plugins.plugin*), 1266
KnowledgeBaseSerializer (*class in angr.angrdb.serializers.kb*), 952
KnownAnalysesPlugin (*class in angr.analyses.analysis*), 275
KnownPattern (*class in angr.analyses.decompiler.known_patterns.pattern*), 520
KnownPatternCallInfo (*class in angr.analyses.decompiler.peephole_optimizations.known_pattern_* 612
KnownPatternFinder (*class in angr.analyses.decompiler.known_patterns.finder*), 509
KnownPatternMatch (*class in angr.analyses.decompiler.known_patterns.finder*), 507
KnownPatternOutliner (*class in angr.analyses.decompiler.optimization_passes.known_pattern_ou* 583
KnownPatternTemplate (*class in angr.analyses.decompiler.known_patterns.templates*), 531
KnownStructs (*class in angr.rust.knowledge_plugins.known_structs*), 1843
KsudField (*class in angr.analyses.decompiler.known_patterns.wdk_shared* 534

## L


label (*angr.analyses.decompiler.known_patterns.gating.AllOf property*), 514
label (*angr.analyses.decompiler.known_patterns.gating.AnyOf property*), 513
label (*angr.analyses.decompiler.known_patterns.gating.CorroboratedBy property*), 513
label (*angr.analyses.decompiler.known_patterns.gating.CorroboratedByPattern property*), 514
label (*angr.analyses.decompiler.known_patterns.gating.PatternGate property*), 512
label (*angr.analyses.decompiler.known_patterns.gating.TargetGate property*), 512
Label (*class in angr.ailment.statement*), 225
Label (*class in angr.analyses.disassembly*), 740
Label (*class in angr.analyses.reassembler*), 870
label_got() (*angr.analyses.reassembler.SymbolManager method*), 871
LabelCollector (*class in angr.analyses.decompiler.label_collector*), 535
LabeledMemory (*class in angr.storage.memory_mixins*), 1622
LabeledPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 1642
LabelMergerMixin (*class in angr.storage.memory_mixins.label_merger_mixin*), 1638
Labels (*class in angr.knowledge_plugins.labels*), 1264
LabelsSerializer (*class in angr.angrdb.serializers.labels*), 952
LambdaAttrIter (*class in angr.state_plugins.history*), 1563
LambdaIterIter (*class in angr.state_plugins.history*), 1563
language (*angr.analyses.decompiler.known_patterns.context.PatternContext property*), 489
language (*angr.analyses.language_detector.LanguageDetectionResult attribute*), 786
language (*angr.analyses.language_detector.LanguageDetector property*), 787
LANGUAGE (*angr.engines.pcode.cc.SimCCM68k attribute*), 1034
LANGUAGE (*angr.engines.pcode.cc.SimCCPARISC attribute*), 1035
LANGUAGE (*angr.engines.pcode.cc.SimCCPCodeBase attribute*), 1034
LANGUAGE (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 1035
LANGUAGE (*angr.engines.pcode.cc.SimCCRISCV attribute*), 1035
LANGUAGE (*angr.engines.pcode.cc.SimCCSH4 attribute*), 1035
LANGUAGE (*angr.engines.pcode.cc.SimCCSPARC attribute*), 1035
LANGUAGE (*angr.engines.pcode.cc.SimCCXtensa attribute*), 1036
language_confidence (*angr.project.Project property*), 1392
language_is_certain (*angr.project.Project property*), 1392
LanguageDetectionConfidenceLevel (*class in angr.analyses.language_detector*), 786
LanguageDetectionResult (*class in angr.analyses.language_detector*), 786
LanguageDetector (*class in angr.analyses.language_detector*), 787
languages (*angr.analyses.decompiler.known_patterns.KnownPatternTempl attribute*), 463
languages (*angr.analyses.decompiler.known_patterns.templates.KnownPa attribute*), 532
languages() (*angr.project.Project method*), 1392
LARGE_SWITCH (*angr.analyses.code_tagging.CodeTags attribute*), 351
last_addr (*angr.storage.memory_object.SimMemoryObject property*), 1667
last_edge_hitmap (*angr.state_plugins.history.SimStateHistory property*), 1562
last_node() (*in module angr.analyses.decompiler.utils*), 723
last_nonlabel_statement() (*in module angr.analyses.decompiler.utils*), 722
last_stmt() (*angr.rust.mixins.cfa_mixin.CFAMixin static method*), 1417
last_time (*angr.procedures.linux_kernel.time.time property*), 1367
lazy_import() (*in module angr.utils.lazy_import*), 1691
lca() (*angr.analyses.typehoon.simple_solver.TypeLattice method*), 899
leaf (*angr.analyses.flirt.flirt_node.FlirtNode property*), 757
LEFT (*angr.analyses.typehoon.simple_solver.ConstraintGraphTag attribute*), 901
length (*angr.state_plugins.sim_action_object.SimActionObject property*), 1586
LengthLimiter (*class in angr.exploration_techniques.lengthlimiter*), 1102
Let (*class in angr.ailment.expression*), 220
LibcHandlers (*class in angr.analyses.reaching_definitions.function_handler_library*), 845
LibcStdioHandlers (*class in angr.analyses.reaching_definitions.function_handler_library.stdio*), 846
LibcStdlibHandlers (*class in*

*angr.analyses.reaching_definitions.function_handler_library.glibc*), 850
LibcStringHandlers (*class in angr.analyses.reaching_definitions.function_handler_library.glibc*), 852
LibcUnistdHandlers (*class in angr.analyses.reaching_definitions.function_handler_library.glibc*), 855
Librust (*class in angr.rust.knowledge_plugins.librust*), 1413
lift() (*angr.analyses.typehoon.translator.TypeTranslator method*), 906
lift() (*angr.engines.ail.engine_light.SimEngineAILSimState method*), 1004
lift() (*angr.engines.light.engine.SimEngineLight method*), 1011
lift() (*angr.engines.pcode.lifter.Lifter method*), 1042
lift() (*angr.engines.pcode.lifter.PcodeBasicBlockLifter method*), 1043
lift() (*in module angr.engines.pcode.lifter*), 1042
lift_addr() (*angr.engines.ail.engine_light.SimEngineAILSimState method*), 1003
lift_pcode() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 1045
lift_soot() (*angr.engines.soot.engine.SootMixin method*), 1047
lift_vex() (*angr.engines.pcode.lifter.PcodeLifterEngineMixin method*), 1044
lift_vex() (*angr.engines.vex.lifter.VEXLifter method*), 1079
Lifter (*class in angr.engines.pcode.lifter*), 1041
likes() (*angr.ailment.Block method*), 210
likes() (*angr.ailment.block.Block method*), 215
likes() (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseHeadStatement method*), 714
lineage (*angr.state_plugins.history.SimStateHistory property*), 1562
lineage() (*angr.state_hierarchy.StateHierarchy method*), 1494
LinkedListNode (*class in angr.utils.graph*), 1687
linux_syscall_update_error_reg() (*angr.calling_conventions.SimCCSyscall method*), 973
LinuxLoader (*class in angr.procedures.linux_loader.sim_loader*), 1368
list_default_plugins() (*angr.misc.plugins.PluginPreset method*), 1319
list_functions() (*in module angr.mcp.server*), 1304
list_next_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 518
list_prev_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 518
list_projects() (*in module angr.mcp.server*), 1307
list_sessions() (*angr.mcp.session.SessionManager method*), 1309
list_variable_names() (*in module angr.analyses.decompiler.edits.resolve*), 448
ListAdd (*class in angr.procedures.java_util.list*), 1350
listen (*class in angr.procedures.posix.listen*), 1373
ListGet (*class in angr.procedures.java_util.list*), 1350
ListGetFirst (*class in angr.procedures.java_util.list*), 1350
ListInit (*class in angr.procedures.java_util.list*), 1350
ListPage (*class in angr.storage.memory_mixins.paged_memory.pages.list_*, 1649
ListPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mix*, 1643
ListPagesWithLabelsMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mix*, 1643
ListSize (*class in angr.procedures.java_util.list*), 1350
live_vars_by_stmt() (*angr.analyses.s_liveness.SLivenessAnalysis method*), 878
LiveDefinitions (*class in angr.analyses.ddg*), 362
LiveDefinitions (*class in angr.knowledge_plugins.key_definitions.live_definitions*), 1250
Liveness (*class in angr.knowledge_plugins.key_definitions.liveness*), 1257
LiveVariables (*class in angr.knowledge_plugins.variables.variable_manager*), 1283
llm_client (*angr.project.Project property*), 1389
llm_refine() (*angr.analyses.decompiler.decompiler.Decompiler method*), 416
llm_suggest_function_name() (*angr.analyses.decompiler.decompiler.Decompiler method*), 416
llm_suggest_variable_names() (*angr.analyses.decompiler.decompiler.Decompiler method*), 416
llm_suggest_variable_types() (*angr.analyses.decompiler.decompiler.Decompiler method*), 416
llm_summarize_function() (*angr.analyses.decompiler.decompiler.Decompiler method*), 417
LLMClient (*class in angr.llm_client*), 1293
Load (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransfo attribute*), 334
Load (*class in angr.ailment.expression*), 220
Load (*class in angr.analyses.typehoon.typevars*), 919

load() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 895
load() (*angr.angrdb.db.AngrDB method*), 942
load() (*angr.angrdb.serializers.bookmarks.BookmarksSerializer static method*), 950
load() (*angr.angrdb.serializers.callgraph.CallGraphSerializer static method*), 950
load() (*angr.angrdb.serializers.cfg_model.CFGModelSerializer static method*), 951
load() (*angr.angrdb.serializers.comments.CommentsSerializer static method*), 951
load() (*angr.angrdb.serializers.funcs.FunctionManagerSerializer static method*), 952
load() (*angr.angrdb.serializers.kb.KnowledgeBaseSerializer static method*), 952
load() (*angr.angrdb.serializers.labels.LabelsSerializer static method*), 953
load() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 954
load() (*angr.angrdb.serializers.structured_code.StructuredCodeManagerSerializer static method*), 955
load() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 956
load() (*angr.angrdb.serializers.xrefs.XRefsSerializer static method*), 956
load() (*angr.procedures.win32.dynamic_loading.LoadLibraryA method*), 1383
load() (*angr.state_plugins.light_registers.SimLightRegisters method*), 1577
load() (*angr.storage.memory_mixins.actions_mixin.ActionsMixin method*), 1633
load() (*angr.storage.memory_mixins.actions_mixin.ActionsMixin method*), 1633
load() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 1634
load() (*angr.storage.memory_mixins.bvv_conversion_mixin.DataNormalizationMixin method*), 1634
load() (*angr.storage.memory_mixins.conditional_store_mixin.ConditionalStoreMixin method*), 1634
load() (*angr.storage.memory_mixins.inspect_mixin.InspectMixin method*), 1636
load() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1636
load() (*angr.storage.memory_mixins.keyvalue_memory_mixin.KeyValueMemoryMixin method*), 1638
load() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1639
load() (*angr.storage.memory_mixins.name_resolution_mixin.NameResolutionMixin method*), 1641
load() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1642
load() (*angr.storage.memory_mixins.paged_memory.pages.base.ListPage method*), 1649
load() (*angr.storage.memory_mixins.paged_memory.pages.list_page.ListPage method*), 1651
load() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 1655
load() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.RegionMetaMixin method*), 1662
load() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 1663
load() (*angr.storage.memory_mixins.simple_interface_mixin.SimpleInterfaceMixin method*), 1664
load() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin method*), 1665
load() (*angr.storage.memory_mixins.size_resolution_mixin.SizeNormalizationMixin method*), 1665
load() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 1665
load() (*angr.storage.memory_mixins.underconstrained_mixin.UnderconstrainedMixin method*), 1666
load() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 1666
load() (*angr.vaults.Vault method*), 1704
load() (*angr.vaults.VaultDirShelf method*), 1705
load_all_definitions() (*in module angr.procedures.definitions*), 1333
load_all_loggers() (*angr.misc.loggers.Loggers method*), 1316
load_all_spilled() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1186
load_all_spilled() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1181
load_all_spilled() (*angr.knowledge_plugins.cfg.spilling_digraph.SpillingDiGraph method*), 1188
load_all_spilled() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1221
load_all_spilled_edges() (*angr.knowledge_plugins.cfg.spilling_digraph.SpillingDiGraph method*), 1188
load_annotations() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1643
LOAD_ARG_BLACKLIST (*angr.angrdb.serializers.loader.LoaderSerializer attribute*), 953
LOAD_ARG_BLACKLIST (*angr.angrdb.serializers.LoaderSerializer attribute*), 948
load_array_element() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1637
load_array_elements() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1637
load_binary() (*in module angr.mcp.server*), 1303
load_dynamic_vars() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 956
load_external_definitions() (*in module angr.procedures.definitions*), 1333
load_field() (*angr.engines.soot.values.thisref.SimSootValue_ThisRef*

*method*), 1062
load_from_dwarf() (*angr.knowledge_plugins.debug_variables.DebugVariableManager method*), 1193
load_from_dwarf() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1289
load_internal() (*angr.angrdb.serializers.variables.VariableManagerSerializer static method*), 956
load_local_variable() (*angr.knowledge_plugins.propagations.states.PropagatorVEXState method*), 1275
load_register() (*angr.knowledge_plugins.propagations.states.PropagatorVEXState method*), 1276
load_shellcode() (*in module angr.project*), 1388
load_signature() (*in module angr.flirt*), 1114
load_signatures() (*in module angr.flirt*), 1114
load_type_collections() (*angr.angrdb.db.AngrDB static method*), 941
load_type_collections() (*in module angr.procedures.definitions*), 1333
load_win32_type_collections() (*in module angr.procedures.definitions*), 1333
load_win32api_definitions() (*in module angr.procedures.definitions*), 1333
load_with_labels() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.LabeledPagesMixin method*), 1642
LoadArgsJSONDecoder (*class in angr.angrdb.serializers.loader*), 953
LoadArgsJSONEncoder (*class in angr.angrdb.serializers.loader*), 953
loaded_type_collection_names() (*angr.angrdb.db.AngrDB static method*), 940
LoaderSerializer (*class in angr.angrdb.serializers.loader*), 953
LoadHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 336
LoadLibrary (*class in angr.procedures.java_lang.load_library*), 1349
LoadLibraryA (*class in angr.procedures.win32.dynamic_loading*), 1383
LoadLibraryExW (*class in angr.procedures.win32.dynamic_loading*), 1383
loads() (*angr.vaults.Vault method*), 1704
LoadSizeRewriter (*class in angr.analyses.decompiler.optimization_passes.determine_load_size*), 565
loc_repr() (*angr.sim_variable.SimVariable method*), 1478
local_runtime_values (*angr.knowledge_plugins.functions.function.Function property*), 1213
local_types (*angr.analyses.decompiler.decompilation_cache.DecompilationCache property*), 443
local_vars (*angr.procedures.java_jni.object_operations.NewObject attribute*), 1347
local_vars (*angr.procedures.linux_loader.sim_loader.IFuncResolver attribute*), 1368
local_vars (*angr.procedures.linux_loader.sim_loader.LinuxLoader attribute*), 1368
local_vars (*angr.procedures.ntdll.exceptions.KiUserExceptionDispatcher attribute*), 1369
local_vars (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1450
local_vars (*angr.SimProcedure attribute*), 190
LOCALE_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 1574
LOCALE_ARRAY (*angr.state_plugins.SimStateLibc attribute*), 1529
LocalLoopSeer (*class in angr.exploration_techniques.local_loop_seer*), 1102
LocalType (*class in angr.protos.variables_pb2*), 1405
LocalVariableTag (*class in angr.knowledge_plugins.key_definitions.tag*), 1261
LocationBase (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 634
locked() (*angr.misc.picklable_lock.PicklableLock method*), 1317
locked() (*angr.misc.picklable_lock.PicklableRLock method*), 1317
Loggers (*class in angr.misc.loggers*), 1316
longest_ail_graph_subseq() (*in module angr.analyses.decompiler.optimization_passes.duplication_reverter*), 572
longest_ail_subseq() (*in module angr.analyses.decompiler.block_similarity*), 399
longest_prefix() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 918
longest_prefix_lookup() (*in module angr.engines.light.engine*), 1011
looks_like_sql() (*in module angr.utils*), 1668
lookup() (*angr.analyses.typehoon.simple_solver.Sketch method*), 900
lookup() (*angr.knowledge_plugins.labels.Labels method*), 1264
lookup() (*angr.state_plugins.filesystem.SimMount method*), 1550
lookup() (*angr.state_plugins.jni_references.SimStateJNIReferences method*), 1574
lookup_defs() (*angr.analyses.ddg.LiveDefinitions method*), 363
lookup_original() (*angr.state_plugins.trace_additions.ChallRespInfo*

*method*), 1597
Loop (*class in angr.analyses.loopfinder*), 791
loop_heads() (*angr.utils.graph.DirectedGraphHelper method*), 1689
loop_nesting_forest() (*angr.utils.graph.GraphUtils static method*), 1687
LoopAnalysis (*class in angr.analyses.loop_analysis.loop_analysis*), 790
LoopBodyAssignmentCollector (*class in angr.analyses.loop_analysis.loop_analysis*), 789
LoopBodyControlStatementCollector (*class in angr.analyses.loop_analysis.loop_analysis*), 789
LoopFinder (*class in angr.analyses.loopfinder*), 791
LoopNode (*class in angr.analyses.decompiler.structurer_nodes*), 712
LoopNodeFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 635
LoopSeer (*class in angr.exploration_techniques.loop_seer*), 1102
LoopSimplifier (*class in angr.analyses.decompiler.region_simplifiers.loop*), 638
LoopUnroller (*class in angr.analyses.loop_unroller.loop_unroller*), 791
LoopVisitor (*class in angr.analyses.forward_analysis.visitors.loop*), 768
LoopVisitor (*class in angr.analyses.loop_analysis.loop_analysis*), 790
LOW (*angr.analyses.language_detector.LanguageDetectionConfidenceLevel attribute*), 786
LoweredSwitchSimplifier (*class in angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier*), 584
lseek (*class in angr.procedures.linux_kernel.lseek*), 1366
LShift (*angr.engines.light.ArithmeticExpression attribute*), 1006
LShift (*angr.engines.light.data.ArithmeticExpression attribute*), 1009
LT (*angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier.CmpOp attribute*), 639

## M

Macro (*class in angr.ailment.expression*), 221
macro_operand() (*in module angr.analyses.decompiler.known_patterns.posix_macros*), 524
magenta (*angr.misc.ansi.BackgroundColor attribute*), 1314
magenta (*angr.misc.ansi.Color attribute*), 1314
MAGIC_CONSTANTS (*in module angr.utils.constants*), 1675
main() (*in module angr.flirt.build_sig*), 1115
main() (*in module angr.procedures.definitions.parse_glibc*), 1333
main() (*in module angr.procedures.definitions.parse_syscalls_from_local_s*), 1334
main() (*in module angr.procedures.definitions.parse_win32json*), 1334
main_executable_region_limbos_contain() (*angr.analyses.reassembler.Reassembler method*), 876
main_executable_regions (*angr.analyses.reassembler.Reassembler property*), 875
main_executable_regions_contain() (*angr.analyses.reassembler.Reassembler method*), 876
main_nonexecutable_region_limbos_contain() (*angr.analyses.reassembler.Reassembler method*), 876
main_nonexecutable_regions (*angr.analyses.reassembler.Reassembler property*), 875
main_nonexecutable_regions_contain() (*angr.analyses.reassembler.Reassembler method*), 876
make() (*angr.sim_type.SimTypePointer method*), 1466
make() (*angr.sim_type.SimTypeReference method*), 1467
MAKE_ARGUMENT_LIST (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
make_breakpoint() (*angr.state_plugins.inspect.SimInspector method*), 1571
make_bv_sizes_equal() (*in module angr.engines.pcode.behavior*), 1014
MAKE_CALL_SITES (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
make_child() (*angr.state_plugins.history.SimStateHistory method*), 1563
make_concrete_int() (*angr.sim_state.SimState method*), 1455
make_continuation() (*angr.sim_procedure.SimProcedure method*), 1450
make_copy() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
make_extern() (*angr.code_location.AILCodeLocation static method*), 986
make_function_codeloc() (*angr.analyses.reaching_definitions.function_handler.FunctionHa method*), 843

make_functions() (*angr.analyses.cfg.cfg_base.CFGBase method*), 310
make_functions() (*angr.analyses.cfg.cfg_fast_soot.CFGFastSoot method*), 325
make_ident() (*angr.storage.file.SimFileBase static method*), 1610
make_initial_state() (*angr.analyses.identifier.identify.Identifier static method*), 785
make_liveness_snapshot() (*angr.knowledge_plugins.key_definitions.liveness.Liveness method*), 1258
make_liveness_snapshot() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 1259
make_ntstatus_severity_template() (*in module angr.analyses.decompiler.known_patterns.wdk_shared_data*), 535
make_operations() (*in module angr.engines.vex.claripy.irop*), 1073
make_phi_node() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1284
make_return_edges() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 325
MAKE_RETURN_SITES (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
make_shared_user_data_template() (*in module angr.analyses.decompiler.known_patterns.wdk_shared_data*), 534
make_std_vector_back_template() (*in module angr.analyses.decompiler.known_patterns.stl_accessors2*), 530
make_std_vector_index_template() (*in module angr.analyses.decompiler.known_patterns.stl_containers*), 531
make_std_vector_size_exactdiv_template() (*in module angr.analyses.decompiler.known_patterns.std_vector_utils*), 529
make_std_vector_size_template() (*in module angr.analyses.decompiler.known_patterns.std_vector_utils*), 528
make_symbolic_state() (*angr.analyses.identifier.identify.Identifier static method*), 785
make_template() (*in module angr.analyses.decompiler.known_patterns.template*), 533
make_type_reference() (*in module angr.utils.types*), 1702
MakeTypecastsImplicit (*class in angr.analyses.decompiler.structured_codegen.c*), 687
MakeTypecastsImplicit (*class in angr.analyses.decompiler.structured_codegen.rust*), 709
malloc (*class in angr.analyses.identifier.functions.malloc*), 774
malloc (*class in angr.procedures.libc.malloc*), 1358
malloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 1558
manager (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 626
MANUAL (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototypeKind attribute*), 1150
ManualMergepoint (*class in angr.exploration_techniques.manual_mergepoint*), 1005
manyargs (*class in angr.procedures.testing.manyargs*), 1380
map() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionData method*), 1661
map_callsites() (*angr.analyses.identifier.identify.Identifier method*), 785
map_offsets_to_bases() (*angr.analyses.typehoon.simple_solver.SimpleSolverTypeVariableMapping method*), 902
map_region() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 1634
map_region() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1639
map_region() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1642
MapContainsKey (*class in angr.procedures.java_util.map*), 1350
MapFail (*angr.protos.primitives_pb2.Edge attribute*), 1403
MapGet (*class in angr.procedures.java_util.map*), 1350
MapInit (*class in angr.procedures.java_util.map*), 1350
MapKeySet (*class in angr.procedures.java_util.map*), 1350
mapped_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 1568
mapped_address (*angr.state_plugins.InspectAttrs attribute*), 1499
mapped_page (*angr.state_plugins.inspect.InspectAttrs attribute*), 1568
mapped_page (*angr.state_plugins.InspectAttrs attribute*), 1499
MapPut (*class in angr.procedures.java_util.map*), 1350
MapSize (*class in angr.procedures.java_util.map*), 1350
mark_const() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 866
mark_dirty() (*angr.knowledge_plugins.functions.function.Function method*), 1214
mark_edge() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 629
mark_function_alignments() (*angr.analyses.cfg.cfg_base.CFGBase*

*method*), 310
mark_guard() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 866
mark_node_addr_has_return() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1167
mark_nofilter() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 1103
mark_nonreturning_calls_endpoints() (*angr.knowledge_plugins.functions.function.Function method*), 1214
mark_okfilter() (*angr.exploration_techniques.manual_mergepoint.ManualMergepoint method*), 1103
marker_to_state() (*angr.analyses.pathfinder.Pathfinder method*), 795
match() (*angr.analyses.decompiler.known_patterns.dsl.PAny method*), 492
match() (*angr.analyses.decompiler.known_patterns.dsl.PAssign method*), 503
match() (*angr.analyses.decompiler.known_patterns.dsl.PatternExpr method*), 492
match() (*angr.analyses.decompiler.known_patterns.dsl.PatternStmt method*), 502
match() (*angr.analyses.decompiler.known_patterns.dsl.PBinOp method*), 494
match() (*angr.analyses.decompiler.known_patterns.dsl.PCall method*), 497
match() (*angr.analyses.decompiler.known_patterns.dsl.PCallResult method*), 498
match() (*angr.analyses.decompiler.known_patterns.dsl.PCallStmt method*), 504
match() (*angr.analyses.decompiler.known_patterns.dsl.PChoice method*), 502
match() (*angr.analyses.decompiler.known_patterns.dsl.PCondJump method*), 504
match() (*angr.analyses.decompiler.known_patterns.dsl.PConst method*), 493
match() (*angr.analyses.decompiler.known_patterns.dsl.PConv method*), 495
match() (*angr.analyses.decompiler.known_patterns.dsl.PDefOf method*), 498
match() (*angr.analyses.decompiler.known_patterns.dsl.PExtract method*), 496
match() (*angr.analyses.decompiler.known_patterns.dsl.PField method*), 499
match() (*angr.analyses.decompiler.known_patterns.dsl.PITE method*), 501
match() (*angr.analyses.decompiler.known_patterns.dsl.PLoad method*), 496
match() (*angr.analyses.decompiler.known_patterns.dsl.PPhi method*), 501
match() (*angr.analyses.decompiler.known_patterns.dsl.PStmtExpr method*), 500
match() (*angr.analyses.decompiler.known_patterns.dsl.PStore method*), 503
match() (*angr.analyses.decompiler.known_patterns.dsl.PUnaryOp method*), 494
match() (*angr.analyses.decompiler.known_patterns.dsl.PVVar method*), 493
match() (*angr.rust.knowledge_plugins.known_structs.StructMatcher method*), 1412
match() (*angr.utils.mergepoint.Mergepoint method*), 725
match_call() (*angr.rust.mixins.cfa_mixin.CFAMixin method*), 1417
match_function() (*angr.analyses.flirt.flirt_matcher.FlirtMatcher method*), 756
match_stmt_classes() (*in module angr.analyses.decompiler.utils*), 725
match_with_known_structs() (*angr.rust.knowledge_plugins.known_structs.KnownStructs method*), 1412
MatchCtx (*class in angr.analyses.decompiler.known_patterns.dsl*), 491
matches() (*angr.analyses.decompiler.block_simplifier.PeepholeOptimizati method*), 400
matches() (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchC method*), 714
matches() (*angr.knowledge_plugins.key_definitions.definition.Definition method*), 1247
matches() (*angr.knowledge_plugins.key_definitions.definition.DefinitionM method*), 1246
MatchState (*class in angr.analyses.decompiler.known_patterns.dsl*), 490
materialize() (*angr.analyses.decompiler.region_overlay.RegionOverlayC method*), 632
max() (*angr.state_plugins.solver.SimSolver method*), 1590
max_allocation (*angr.state_plugins.cgc.SimStateCGC attribute*), 1545
max_allocation (*angr.state_plugins.SimStateCGC attribute*), 1525
MAX_BYTES (*angr.analyses.cfg.cfb.Unknown attribute*), 304
MAX_DATA_REFS (*angr.engines.pcode.lifter.IRSB attribute*), 1038
MAX_EXITS (*angr.engines.pcode.lifter.IRSB attribute*), 1038
max_gap (*angr.analyses.decompiler.known_patterns.dsl.PStmtSeq attribute*), 505
max_gap (*angr.analyses.decompiler.known_patterns.PStmtSeq attribute*), 475
max_int() (*angr.state_plugins.solver.SimSolver method*), 1595
MAX_LIFTED_EXPR_NODES (*angr.analyses.decompiler.known_patterns.finder.KnownPatternF attribute*), 510
MAX_LIFTED_EXPR_NODES (*angr.analyses.decompiler.known_patterns.KnownPatternFinder attribute*), 461

MAX_LIFTED_EXPR_NODES (*angr.analyses.decompiler.KnownPatternFinder attribute*), 382
max_null_index (*angr.procedures.libc.strlen.strlen attribute*), 1360
MAX_ONE_CALL (*angr.analyses.decompiler.structuring.phoenix.MultiStmtExprMode attribute*), 717
MAX_REMOTE_CHASE_DEPTH (*angr.analyses.decompiler.known_patterns.finder.KnownPatternFinder attribute*), 509
MAX_REMOTE_CHASE_DEPTH (*angr.analyses.decompiler.known_patterns.KnownPatternFinder attribute*), 459
MAX_REMOTE_CHASE_DEPTH (*angr.analyses.decompiler.KnownPatternFinder attribute*), 381
MAX_ROUNDS (*angr.analyses.decompiler.optimization_passes.engine_base.OptimizationPassStructurer attribute*), 584
MAX_ROUNDS (*angr.analyses.decompiler.optimization_passes.expr_opt.ExprOptimization attribute*), 551
MAX_SHARED_NAME (*angr.analyses.decompiler.semantic_naming.SemanticNaming attribute*), 650
MAX_SHARED_NAME (*angr.analyses.decompiler.semantic_naming.SemanticNamingOrchestrator attribute*), 646
max_tv_id (*angr.analyses.typehoon.typevars.TypeVariableManager property*), 916
max_vvar_id (*angr.analyses.decompiler.ssailification.rewriting_engine.SsailificationRewritingEngine property*), 653
maximize_similarity_of_blocks() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.DuplicationReverter method*), 571
maximum_syscall_number() (*angr.procedures.definitions.SimSyscallLibrary method*), 1330
MCPAngrError, 1301
MEDIUM (*angr.analyses.language_detector.LanguageDetectionConfidenceLevel attribute*), 786
meet() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 904
meet_simtypes() (*angr.analyses.typehoon.simple_solver.SimpleSolver class method*), 904
mem (*angr.state_plugins.debug_variables.SimDebugVariable property*), 1545
MEM (*angr.state_plugins.sim_action.SimAction attribute*), 1584
MEM (*angr.state_plugins.SimAction attribute*), 1503
mem() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 1243
mem_concrete() (*angr.sim_state.SimState method*), 1455
mem_load() (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier method*), 580
mem_map() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 1603
mem_map_ptr() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 1603
MEM_PATCH (*class in angr.state_plugins.unicorn_engine*), 1599
MEM_READ (*angr.state_plugins.inspect.EventType attribute*), 1565
mem_read_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_read_address (*angr.state_plugins.InspectAttrs attribute*), 1497
mem_read_condition (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_read_condition (*angr.state_plugins.InspectAttrs attribute*), 1497
mem_read_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_read_endness (*angr.state_plugins.InspectAttrs attribute*), 1498
mem_read_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_read_expr (*angr.state_plugins.InspectAttrs attribute*), 1497
mem_read_length (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_read_length (*angr.state_plugins.InspectAttrs attribute*), 1497
mem_reset() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 1603
mem_store() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.DuplicationReverter method*), 571
mem_unmap() (*angr.state_plugins.unicorn_engine.Uniwrapper method*), 1603
mem_untyped (*angr.state_plugins.debug_variables.SimDebugVariable property*), 1545
MEM_WRITE (*angr.state_plugins.inspect.EventType attribute*), 1565
mem_write_address (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_write_address (*angr.state_plugins.InspectAttrs attribute*), 1498
mem_write_condition (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_write_condition (*angr.state_plugins.InspectAttrs attribute*), 1498
mem_write_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_write_endness (*angr.state_plugins.InspectAttrs attribute*), 1498
mem_write_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
mem_write_expr (*angr.state_plugins.InspectAttrs attribute*), 1498
mem_write_length (*angr.state_plugins.inspect.InspectAttrs*

*attribute*), 1566
mem_write_length (*angr.state_plugins.InspectAttrs attribute*), 1498
member() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 1546
member() (*angr.state_plugins.view.SimMemView method*), 1607
member_view (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 632
members (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 626
members (*angr.sim_type.SimCppClass property*), 1475
memccpy (*class in angr.procedures.libc.memccpy*), 1358
memchr (*class in angr.procedures.libc.memchr*), 1358
memcmp (*class in angr.analyses.identifier.functions.memcmp*), 775
memcmp (*class in angr.procedures.libc.memcmp*), 1358
memcpy (*class in angr.analyses.identifier.functions.memcpy*), 775
memcpy (*class in angr.procedures.libc.memcpy*), 1358
MemDepNode (*class in angr.analyses.data_dep.dep_nodes*), 359
memmove (*class in angr.procedures.libc.memmove*), 1358
memo() (*angr.state_plugins.plugin.SimStatePlugin static method*), 1578
memoize() (*in module angr.analyses.typehoon.typeconsts*), 906
Memory (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 359
Memory (*angr.analyses.data_dep.DepNodeTypes attribute*), 356
MEMORY (*angr.analyses.reaching_definitions.AtomKind attribute*), 806
memory (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
MEMORY (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 1241
MEMORY (*angr.knowledge_plugins.variables.variable_manager.VariableType attribute*), 1283
MEMORY (*angr.knowledge_plugins.variables.VariableType attribute*), 1281
memory() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 1243
memory_args (*angr.calling_conventions.SimCC property*), 968
memory_data (*angr.analyses.cfg.cfg_fast.CFGFast property*), 325
MEMORY_ERROR (*angr.emulator.EmulatorStopReason attribute*), 996
MEMORY_ERROR (*angr.EmulatorStopReason attribute*), 161
MEMORY_PAGE_MAP (*angr.state_plugins.inspect.EventType attribute*), 1565
memory_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
MemoryData (*class in angr.knowledge_plugins.cfg.memory_data*), 1178
MemoryData (*class in angr.protos.cfg_pb2*), 1395
MemoryDataSort (*class in angr.knowledge_plugins.cfg.memory_data*), 1177
MemoryLoadResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.memload_resolver*), 338
MemoryLocation (*class in angr.knowledge_plugins.key_definitions.atoms*), 1244
MemoryMappingError, 1602
MemoryMixin (*class in angr.storage.memory_mixins.memory_mixin*), 1638
MemoryObjectMixin (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 1647
MemoryObjectSetMixin (*class in angr.storage.memory_mixins.paged_memory.pages.cooperation*), 1648
MemoryOperand (*class in angr.analyses.disassembly*), 742
MemoryRegion (*class in angr.analyses.cfg.cfb*), 304
MemoryRegionMetaMixin (*class in angr.storage.memory_mixins.regioned_memory.region_meta_mix*), 1662
MemoryValue (*class in angr.state_plugins.unicorn_engine*), 1599
MemoryVariable (*class in angr.protos.variables_pb2*), 1405
MemoryWatcher (*class in angr.exploration_techniques.memory_watcher*), 1103
MemoryWriteCollector (*class in angr.rust.analyses.rust_calling_convention.fact_collector*), 1408
mempcpy (*class in angr.procedures.libc.mempcpy*), 1358
memset (*class in angr.analyses.identifier.functions.memset*), 776
memset (*class in angr.procedures.libc.memset*), 1358
merge() (*angr.analyses.decompiler.edits.results.Refresh method*), 449
merge() (*angr.analyses.decompiler.optimization_passes.engine_base.Simp method*), 574
merge() (*angr.analyses.decompiler.ssailification.traversal_state.TraversalS method*), 659
merge() (*angr.analyses.decompiler.stackarg_offset_manager.StackArgOffs method*), 660
merge() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsS method*), 863
merge() (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTrackerS*

*method*), 894
merge() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 895
merge() (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery method*), 926
merge() (*angr.analyses.variable_recovery.variable_recovery_fast.VariableRecoveryFast method*), 930
merge() (*angr.concretization_strategies.base.SimConcretizationStrategy method*), 991
merge() (*angr.keyed_region.KeyedRegion method*), 1116
merge() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 1174
merge() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 1248
merge() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1252
merge() (*angr.knowledge_plugins.key_definitions.rd_model.ReachingDefinitionsModel method*), 1260
merge() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1263
merge() (*angr.knowledge_plugins.propagations.states.PropagatorState method*), 1274
merge() (*angr.knowledge_plugins.propagations.states.PropagatorVEXState method*), 1275
merge() (*angr.sim_manager.SimulationManager method*), 1447
merge() (*angr.sim_state.SimState method*), 1455
merge() (*angr.state_plugins.edge_hitmap.SimStateEdgeHitmap method*), 1547
merge() (*angr.state_plugins.plugin.SimStatePlugin method*), 1578
merge() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1649
merge() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 1662
merge_candidate_keys() (*in module angr.utils.cowdict*), 1676
merge_edge_data() (*in module angr.analyses.decompiler.region_overlay*), 624
merge_jobs() (*angr.analyses.vfg.CallAnalysis method*), 934
merge_overlapping_register_spans() (*in module angr.analyses.calling_convention.utils*), 289
merge_points() (*angr.annocfg.AnnotatedCFG method*), 958
merge_replacements() (*angr.knowledge_plugins.propagations.states.PropagatorState static method*), 1273
merge_to_top() (*angr.keyed_region.KeyedRegion method*), 1116
merge_transitions() (*in module angr.analyses.cfg_slice_to_sink.transitions*), 351
merge_two_split_type() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.DuplicationReverter method*), 589
merged_jobs (*angr.analyses.forward_analysis.job_info.JobInfo property*), 764
MergedFastState (*class in angr.procedures.win_user32.messagebox*), 1388
MessageBoxA (*class in angr.procedures.win_user32.messagebox*), 1388
MessageBoxExA (*class in angr.procedures.win_user32.messagebox*), 1388
MessageBoxExW (*class in angr.procedures.win_user32.messagebox*), 1388
MessageBoxIndirectA (*class in angr.procedures.win_user32.messagebox*), 1388
MessageBoxIndirectW (*class in angr.procedures.win_user32.messagebox*), 1388
MessageBoxW (*class in angr.procedures.win_user32.messagebox*), 1388
MicrosoftAMD64ArgSession (*class in angr.calling_conventions*), 973
min() (*angr.state_plugins.solver.SimSolver method*), 1590
MIN_ASSIGNMENTS (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 578
MIN_ASSIGNMENTS (*angr.analyses.decompiler.optimization_passes.InlinedStringTransformationSimplifier attribute*), 548
minimum_int() (*angr.state_plugins.solver.SimSolver method*), 1595
minimum_syscall_number() (*angr.procedures.definitions.SimSyscallLibrary method*), 1330
mips_dirtyhelper_rdhwr() (*in module angr.engines.vex.heavy.dirty*), 1078
MipsElfFastResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast*), 338
MipsElfGotResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.mips_elf_got*), 339
MIPSGPHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 337
MipsGpSettingSimplifier (*class in angr.analyses.decompiler.optimization_passes.mips_gp_setting_simplifier*), 585
misses (*angr.analyses.pathfinder.SimStateMarker attribute*), 794
MISSING (*angr.analyses.pathfinder.SuccessorsKind attribute*), 794

MissingMemoryError, 1648
MixedPermissonsError, 1603
mmap (*class in angr.procedures.posix.mmap*), 1373
mmap2 (*class in angr.procedures.linux_kernel.mmap*), 1366
mnemonic (*angr.analyses.disassembly.Instruction property*), 741
mnemonic (*angr.block.CapstoneInsn property*), 960
mnemonic (*angr.block.DisassemblerInsn property*), 959
mnemonic (*angr.block.PCodeInsn property*), 960
mnemonic (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 1037
mode_to_flag() (*in module angr.procedures.libc.fopen*), 1354
mode_to_flag() (*in module angr.procedures.posix.fdopen*), 1371
model (*angr.analyses.cfg.cfg_base.CFGBase property*), 308
model_config (*angr.llm_models.FunctionNameSuggestion attribute*), 1295
model_config (*angr.llm_models.VariableNameSuggestions attribute*), 1295
model_config (*angr.llm_models.VariableRename attribute*), 1295
model_config (*angr.llm_models.VariableTypeChange attribute*), 1295
model_config (*angr.llm_models.VariableTypeSuggestions attribute*), 1295
ModSimplifier (*class in angr.analyses.decompiler.optimization_passes.mod_simplifier*), 586
ModSimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.mod_simplifier*), 586
module
    angr, 155
    angr.ail_callable, 208
    angr.ailment, 208
    angr.ailment.block, 214
    angr.ailment.block_walker, 215
    angr.ailment.constant, 216
    angr.ailment.converter_common, 216
    angr.ailment.converter_pcode, 217
    angr.ailment.converter_vex, 217
    angr.ailment.expression, 217
    angr.ailment.manager, 224
    angr.ailment.statement, 224
    angr.ailment.tagged_object, 226
    angr.ailment.utils, 227
    angr.analyses, 228
    angr.analyses.analysis, 275
    angr.analyses.backward_slice, 277
    angr.analyses.binary_optimizer, 279
    angr.analyses.bindiff, 280
    angr.analyses.boyscout, 283
    angr.analyses.callee_cleanup_finder, 283
    angr.analyses.calling_convention, 283
    angr.analyses.calling_convention.calling_convention, 285
    angr.analyses.calling_convention.fact_collector, 286
    angr.analyses.calling_convention.utils, 289
    angr.analyses.cdg, 289
    angr.analyses.cfg, 290
    angr.analyses.cfg.cfb, 304
    angr.analyses.cfg.cfg, 306
    angr.analyses.cfg.cfg_arch_options, 307
    angr.analyses.cfg.cfg_base, 307
    angr.analyses.cfg.cfg_emulated, 310
    angr.analyses.cfg.cfg_fast, 315
    angr.analyses.cfg.cfg_fast_soot, 325
    angr.analyses.cfg.cfg_job_base, 326
    angr.analyses.cfg.go_prologue, 326
    angr.analyses.cfg.indirect_jump_resolvers, 327
    angr.analyses.cfg.indirect_jump_resolvers.aarch64_mach 331
    angr.analyses.cfg.indirect_jump_resolvers.amd64_elf_go 331
    angr.analyses.cfg.indirect_jump_resolvers.amd64_pe_iat 332
    angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast 332
    angr.analyses.cfg.indirect_jump_resolvers.const_resolv 332
    angr.analyses.cfg.indirect_jump_resolvers.constant_val 333
    angr.analyses.cfg.indirect_jump_resolvers.default_reso 333
    angr.analyses.cfg.indirect_jump_resolvers.jumptable, 333
    angr.analyses.cfg.indirect_jump_resolvers.memload_reso 338
    angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fas 338
    angr.analyses.cfg.indirect_jump_resolvers.mips_elf_got 339
    angr.analyses.cfg.indirect_jump_resolvers.propagator_u 339
    angr.analyses.cfg.indirect_jump_resolvers.resolver, 340
    angr.analyses.cfg.indirect_jump_resolvers.syscall_reso 340
    angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_ 341
    angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat, 341

angr.analyses.cfg.meta_structs, 341
angr.analyses.cfg.pe_msvc_eh_structs, 342
angr.analyses.cfg_slice_to_sink, 346
angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink, 348
angr.analyses.cfg_slice_to_sink.graph, 350
angr.analyses.cfg_slice_to_sink.transitions, 351
angr.analyses.class_identifier, 351
angr.analyses.code_tagging, 351
angr.analyses.codecave, 352
angr.analyses.complete_calling_conventions, 353
angr.analyses.congruency_check, 354
angr.analyses.data_dep, 355
angr.analyses.data_dep.data_dependency_analysis, 358
angr.analyses.data_dep.dep_nodes, 359
angr.analyses.data_dep.sim_act_location, 361
angr.analyses.datagraph_meta, 361
angr.analyses.ddg, 361
angr.analyses.decompiler, 367
angr.analyses.decompiler.ail_simplifier, 396
angr.analyses.decompiler.ailgraph_walker, 398
angr.analyses.decompiler.block_io_finder, 398
angr.analyses.decompiler.block_similarity, 398
angr.analyses.decompiler.block_simplifier, 399
angr.analyses.decompiler.block_walkers, 401
angr.analyses.decompiler.callsite_maker, 401
angr.analyses.decompiler.ccall_rewriters, 402
angr.analyses.decompiler.ccall_rewriters.amd64_ccalls, 402
angr.analyses.decompiler.ccall_rewriters.arm_ccalls, 402
angr.analyses.decompiler.ccall_rewriters.rewriter_base, 402
angr.analyses.decompiler.ccall_rewriters.x86_ccalls, 403
angr.analyses.decompiler.clinic, 403
angr.analyses.decompiler.condition_processor, 408
angr.analyses.decompiler.counters, 410
angr.analyses.decompiler.counters.boolean_counter, 411
angr.analyses.decompiler.counters.call_counter, 411
angr.analyses.decompiler.counters.expression_counters, 412
angr.analyses.decompiler.counters.seq_cf_structure_cou 412
angr.analyses.decompiler.decompilation_cache, 412
angr.analyses.decompiler.decompilation_options, 413
angr.analyses.decompiler.decompiler, 414
angr.analyses.decompiler.dephication, 417
angr.analyses.decompiler.dephication.dephication_base, 418
angr.analyses.decompiler.dephication.graph_dephication 419
angr.analyses.decompiler.dephication.graph_rewriting, 419
angr.analyses.decompiler.dephication.graph_vvar_mappin 419
angr.analyses.decompiler.dephication.rewriting_engine, 419
angr.analyses.decompiler.dephication.seqnode_dephicati 420
angr.analyses.decompiler.dirty_rewriters, 421
angr.analyses.decompiler.dirty_rewriters.amd64_dirty, 421
angr.analyses.decompiler.dirty_rewriters.rewriter_base 421
angr.analyses.decompiler.edits, 421
angr.analyses.decompiler.edits.cache, 435
angr.analyses.decompiler.edits.errors, 437
angr.analyses.decompiler.edits.hooks, 438
angr.analyses.decompiler.edits.ops, 442
angr.analyses.decompiler.edits.resolve, 446
angr.analyses.decompiler.edits.results, 449
angr.analyses.decompiler.empty_node_remover, 450
angr.analyses.decompiler.expression_narrower, 451
angr.analyses.decompiler.goto_manager, 452
angr.analyses.decompiler.jump_target_collector, 453
angr.analyses.decompiler.jumptable_entry_condition_rew 453
angr.analyses.decompiler.known_patterns, 453
angr.analyses.decompiler.known_patterns.apply, 485

angr.analyses.decompiler.known_patterns.block_append, 486
angr.analyses.decompiler.known_patterns.containing_memset, 487
angr.analyses.decompiler.known_patterns.context, 487
angr.analyses.decompiler.known_patterns.ctype_table, 489
angr.analyses.decompiler.known_patterns.dsl, 490
angr.analyses.decompiler.known_patterns.finder, 507
angr.analyses.decompiler.known_patterns.gating, 510
angr.analyses.decompiler.known_patterns.generator, 515
angr.analyses.decompiler.known_patterns.kernel_err, 516
angr.analyses.decompiler.known_patterns.layouts, 517
angr.analyses.decompiler.known_patterns.libm_bits, 518
angr.analyses.decompiler.known_patterns.linked_list, 518
angr.analyses.decompiler.known_patterns.pattern, 519
angr.analyses.decompiler.known_patterns.posix_macros, 523
angr.analyses.decompiler.known_patterns.protobuf_hasbits, 524
angr.analyses.decompiler.known_patterns.registry, 524
angr.analyses.decompiler.known_patterns.std_string_cstr, 526
angr.analyses.decompiler.known_patterns.std_string_ctor, 526
angr.analyses.decompiler.known_patterns.std_string_internals, 527
angr.analyses.decompiler.known_patterns.std_string_length, 527
angr.analyses.decompiler.known_patterns.std_swap, 527
angr.analyses.decompiler.known_patterns.std_vector_size, 528
angr.analyses.decompiler.known_patterns.stl_accessors2, 530
angr.analyses.decompiler.known_patterns.stl_containers, 531
angr.analyses.decompiler.known_patterns.templates, 531
angr.analyses.decompiler.known_patterns.vector_clamp, 533
angr.analyses.decompiler.known_patterns.vector_math, 534
angr.analyses.decompiler.known_patterns.wdk_shared_dat 534
angr.analyses.decompiler.label_collector, 535
angr.analyses.decompiler.node_replacer, 535
angr.analyses.decompiler.notes, 535
angr.analyses.decompiler.notes.decompilation_note, 538
angr.analyses.decompiler.notes.deobfuscated_strings, 539
angr.analyses.decompiler.optimization_pass_registry, 540
angr.analyses.decompiler.optimization_passes, 540
angr.analyses.decompiler.optimization_passes.base_ptr_ 559
angr.analyses.decompiler.optimization_passes.call_stmt 560
angr.analyses.decompiler.optimization_passes.code_moti 560
angr.analyses.decompiler.optimization_passes.condition 562
angr.analyses.decompiler.optimization_passes.const_der 562
angr.analyses.decompiler.optimization_passes.const_pro 563
angr.analyses.decompiler.optimization_passes.cross_jum 564
angr.analyses.decompiler.optimization_passes.deadblock 565
angr.analyses.decompiler.optimization_passes.determine 565
angr.analyses.decompiler.optimization_passes.div_simpl 565
angr.analyses.decompiler.optimization_passes.duplicati 566
angr.analyses.decompiler.optimization_passes.duplicati 568
angr.analyses.decompiler.optimization_passes.duplicati 570
angr.analyses.decompiler.optimization_passes.duplicati 571
angr.analyses.decompiler.optimization_passes.duplicati 572
angr.analyses.decompiler.optimization_passes.duplicati 572
angr.analyses.decompiler.optimization_passes.eager_std 573
angr.analyses.decompiler.optimization_passes.eager_std 574
angr.analyses.decompiler.optimization_passes.engine_ba 574
angr.analyses.decompiler.optimization_passes.expr_op_s

angr.analyses.decompiler.optimization_passes.fargc_analyses.decompiler.optimization_passes.win_stack
angr.analyses.decompiler.optimization_passes.inlined_analyses.decompiler.optimization_passes.x86_gcc_g
angr.analyses.decompiler.optimization_passes.inlined_analyses.decompiler.peephole_optimizations,
angr.analyses.decompiler.optimization_passes.inlined_analyses.decompiler.peephole_optimizations.a_div_
angr.analyses.decompiler.optimization_passes.inlined_analyses.decompiler.peephole_optimizations.a_mul_
angr.analyses.decompiler.optimization_passes.inlined_analyses.decompiler.peephole_optimizations.a_mul_
angr.analyses.decompiler.optimization_passes.inlined_analyses.decompiler.peephole_optimizations.a_shl_
angr.analyses.decompiler.optimization_passes.irexpr_analyses.decompiler.peephole_optimizations.a_sub_
angr.analyses.decompiler.optimization_passes.irexpr_analyses.decompiler.peephole_optimizations.a_sub_
angr.analyses.decompiler.optimization_passes.known_analyses.decompiler.peephole_optimizations.a_sub_
angr.analyses.decompiler.optimization_passes.loweranalyses.decompiler.peephole_optimizations.arm_cm
angr.analyses.decompiler.optimization_passes.mipsanalyses.decompiler.peephole_optimizations.base,
angr.analyses.decompiler.optimization_passes.modsanalyses.decompiler.peephole_optimizations.basepo
angr.analyses.decompiler.optimization_passes.optimizanalyses.decompiler.peephole_optimizations.basepo
angr.analyses.decompiler.optimization_passes.peephanalyses.decompiler.peephole_optimizations.bitwis
angr.analyses.decompiler.optimization_passes.registanalyses.decompiler.peephole_optimizations.bitwis
angr.analyses.decompiler.optimization_passes.registanalyses.decompiler.peephole_optimizations.bool_e
angr.analyses.decompiler.optimization_passes.retaddranalyses.decompiler.peephole_optimizations.bswap,
angr.analyses.decompiler.optimization_passes.retdeanalyses.decompiler.peephole_optimizations.bt_rfl
angr.analyses.decompiler.optimization_passes.returnanalyses.decompiler.peephole_optimizations.cas_in
angr.analyses.decompiler.optimization_passes.returnanalyses.decompiler.peephole_optimizations.cmp_ma
angr.analyses.decompiler.optimization_passes.returnanalyses.decompiler.peephole_optimizations.cmp_su
angr.analyses.decompiler.optimization_passes.stackanalyses.decompiler.peephole_optimizations.cmpord
angr.analyses.decompiler.optimization_passes.staticanalyses.decompiler.peephole_optimizations.coales
angr.analyses.decompiler.optimization_passes.switchanalyses.decompiler.peephole_optimizations.coales
angr.analyses.decompiler.optimization_passes.varrecanalyses.decompiler.peephole_optimizations.concat
angr.analyses.decompiler.optimization_passes.varslanalyses.decompiler.peephole_optimizations.consta

610 angr.analyses.decompiler.peephole_optimizations.rewrite_a_sub_to_a_sub_const_conversions
610 angr.analyses.decompiler.peephole_optimizations.rewrite_a_xor_a_zero
610 angr.analyses.decompiler.peephole_optimizations.rewrite_addr_sub_to_sub
611 angr.analyses.decompiler.peephole_optimizations.rewrite_cmp_sub_to_cmp
611 angr.analyses.decompiler.peephole_optimizations.rewrite_eax_eax_mod_const_to_ecx_const
612 angr.analyses.decompiler.peephole_optimizations.rewrite_mov_to_xor
612 angr.analyses.decompiler.peephole_optimizations.rewrite_movzx_movsx_to_cast
612 angr.analyses.decompiler.peephole_optimizations.rewrite_mul_to_add
612 angr.analyses.decompiler.peephole_optimizations.rewrite_mul_to_shl
612 angr.analyses.decompiler.peephole_optimizations.rewrite_or_to_add
613 angr.analyses.decompiler.peephole_optimizations.rewrite_shr_to_sar
613 angr.analyses.decompiler.peephole_optimizations.rewrite_x_sub_x_to_zero
613 angr.analyses.decompiler.peephole_optimizations.rewrite_xor_to_not
614 angr.analyses.decompiler.peephole_optimizations.remove_const_insert
614 angr.analyses.decompiler.peephole_optimizations.remove_cxx_destructor_calls
614 angr.analyses.decompiler.peephole_optimizations.remove_empty_if_body
614 angr.analyses.decompiler.peephole_optimizations.remove_noop_conversions
615 angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks
615 angr.analyses.decompiler.peephole_optimizations.remove_redundant_conversions
615 angr.analyses.decompiler.peephole_optimizations.remove_redundant_derefs
616 angr.analyses.decompiler.peephole_optimizations.remove_redundant_insert
616 angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_branch
616 angr.analyses.decompiler.peephole_optimizations.remove_redundant_ite_comparisons
616 angr.analyses.decompiler.peephole_optimizations.remove_redundant_nots
617 angr.analyses.decompiler.peephole_optimizations.remove_redundant_reinterprets
617 angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts
617 angr.analyses.decompiler.peephole_optimizations.remove_redundant_shifts_around_comparators
618 angr.analyses.decompiler.peephole_optimizations.rewrite_a_sub_to_a_sub_const_conversions
618 angr.analyses.decompiler.peephole_optimizations.rewrite_a_xor_a_zero
618 angr.analyses.decompiler.peephole_optimizations.rewrite_addr_sub_to_sub
619 angr.analyses.decompiler.peephole_optimizations.rewrite_cmp_sub_to_cmp
619 angr.analyses.decompiler.peephole_optimizations.rewrite_eax_eax_mod_const_to_ecx_const
619 angr.analyses.decompiler.peephole_optimizations.rewrite_mov_to_xor
620 angr.analyses.decompiler.peephole_optimizations.rewrite_movzx_movsx_to_cast
620 angr.analyses.decompiler.peephole_optimizations.rewrite_mul_to_add
620 angr.analyses.decompiler.peephole_optimizations.rewrite_mul_to_shl
620 angr.analyses.decompiler.peephole_optimizations.rewrite_or_to_add
621 angr.analyses.decompiler.peephole_optimizations.rewrite_shr_to_sar
621 angr.analyses.decompiler.peephole_optimizations.rewrite_x_sub_x_to_zero
621 angr.analyses.decompiler.peephole_optimizations.rewrite_xor_to_not
621 angr.analyses.decompiler.peephole_optimizations.rol_rol_const_to_ror
619 angr.analyses.decompiler.peephole_optimizations.sar_to_signed_mul
620 angr.analyses.decompiler.peephole_optimizations.shl_to_mul
620 angr.analyses.decompiler.peephole_optimizations.simplify_cascading_conversions
620 angr.analyses.decompiler.peephole_optimizations.single_bit_to_boolean
620 angr.analyses.decompiler.peephole_optimizations.single_bit_tmp_simplifier
621 angr.analyses.decompiler.peephole_optimizations.tidy_stack_addr
621 angr.analyses.decompiler.peephole_optimizations.utils
621 angr.analyses.decompiler.presets
angr.analyses.decompiler.presets.basic
angr.analyses.decompiler.presets.fast
angr.analyses.decompiler.presets.full
angr.analyses.decompiler.presets.malware
angr.analyses.decompiler.presets.preset
angr.analyses.decompiler.redundant_label_remover
angr.analyses.decompiler.region_identifier
angr.analyses.decompiler.region_overlay
angr.analyses.decompiler.region_simplifiers
angr.analyses.decompiler.region_simplifiers.cascading_cond_transformer
angr.analyses.decompiler.region_simplifiers.cascading_ifs
angr.analyses.decompiler.region_simplifiers.expr_folding
angr.analyses.decompiler.region_simplifiers.goto
angr.analyses.decompiler.region_simplifiers.if_

angr.analyses.decompiler.region_simplifiers.ifelse, 638
angr.analyses.decompiler.region_simplifiers.loop, 638
angr.analyses.decompiler.region_simplifiers.node_address_finder, 638
angr.analyses.decompiler.region_simplifiers.region_simplifier, 638
angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier, 639
angr.analyses.decompiler.region_simplifiers.switch_expr_simplifier, 641
angr.analyses.decompiler.region_walker, 642
angr.analyses.decompiler.return_maker, 642
angr.analyses.decompiler.semantic_naming, 642
angr.analyses.decompiler.semantic_naming.array_index_naming, 647
angr.analyses.decompiler.semantic_naming.boolean_naming, 647
angr.analyses.decompiler.semantic_naming.call_result_naming, 648
angr.analyses.decompiler.semantic_naming.naming_base, 648
angr.analyses.decompiler.semantic_naming.orchestrator, 650
angr.analyses.decompiler.semantic_naming.pointer_naming, 651
angr.analyses.decompiler.semantic_naming.region_io_analyzer, 651
angr.analyses.decompiler.semantic_naming.size_naming, 652
angr.analyses.decompiler.seq_to_blocks, 652
angr.analyses.decompiler.sequence_walker, 652
angr.analyses.decompiler.ssailification, 652
angr.analyses.decompiler.ssailification.consts, 653
angr.analyses.decompiler.ssailification.rewriting, 653
angr.analyses.decompiler.ssailification.rewriting_engine, 654
angr.analyses.decompiler.ssailification.rewriting_state, 654
angr.analyses.decompiler.ssailification.ssailification, 655
angr.analyses.decompiler.ssailification.traversal, 655
angr.analyses.decompiler.ssailification.traversal_engine, 655
angr.analyses.decompiler.ssailification.traversal_stat 658
angr.analyses.decompiler.stack_item, 659
angr.analyses.decompiler.stackarg_offset_manager, 660
angr.analyses.decompiler.stl_field_accessors, 661
angr.analyses.decompiler.structured_codegen, 661
angr.analyses.decompiler.structured_codegen.base, 668
angr.analyses.decompiler.structured_codegen.c, 670
angr.analyses.decompiler.structured_codegen.c_serializ 688
angr.analyses.decompiler.structured_codegen.dummy, 691
angr.analyses.decompiler.structured_codegen.dwarf_impo 694
angr.analyses.decompiler.structured_codegen.rust, 698
angr.analyses.decompiler.structurer_nodes, 710
angr.analyses.decompiler.structuring, 714
angr.analyses.decompiler.structuring.dream, 716
angr.analyses.decompiler.structuring.phoenix, 717
angr.analyses.decompiler.structuring.recursive_structu 718
angr.analyses.decompiler.structuring.sailr, 718
angr.analyses.decompiler.structurer_base, 719
angr.analyses.decompiler.utils, 720
angr.analyses.decompiler.variable_map, 727
angr.analyses.deobfuscator, 731
angr.analyses.deobfuscator.api_obf_finder, 734
angr.analyses.deobfuscator.api_obf_peephole_optimizer, 735
angr.analyses.deobfuscator.api_obf_type2_finder, 736
angr.analyses.deobfuscator.data_transformation_embedde 737
angr.analyses.deobfuscator.hash_lookup_api_deobfuscato 737
angr.analyses.deobfuscator.irsb_reg_collector, 737
angr.analyses.deobfuscator.scope_ops_analyzer, 738
angr.analyses.deobfuscator.string_obf_finder, 738

angr.analyses.deobfuscator.string_obf_opt_pass, 739
angr.analyses.deobfuscator.string_obf_peephole_optimizers, 739
angr.analyses.disassembly, 739
angr.analyses.disassembly_utils, 743
angr.analyses.dominance_frontier, 743
angr.analyses.fcp, 744
angr.analyses.fcp.fcp, 744
angr.analyses.find_objects_static, 746
angr.analyses.flirt, 747
angr.analyses.flirt.consts, 751
angr.analyses.flirt.flirt, 755
angr.analyses.flirt.flirt_function, 756
angr.analyses.flirt.flirt_matcher, 756
angr.analyses.flirt.flirt_module, 757
angr.analyses.flirt.flirt_node, 757
angr.analyses.flirt.flirt_sig, 758
angr.analyses.flirt.flirt_utils, 761
angr.analyses.forward_analysis, 761
angr.analyses.forward_analysis.forward_analysis, 763
angr.analyses.forward_analysis.job_info, 764
angr.analyses.forward_analysis.visitors, 765
angr.analyses.forward_analysis.visitors.call_graph, 766
angr.analyses.forward_analysis.visitors.function_graph, 766
angr.analyses.forward_analysis.visitors.graph, 766
angr.analyses.forward_analysis.visitors.loop, 768
angr.analyses.forward_analysis.visitors.single_node_graph, 769
angr.analyses.identifier, 769
angr.analyses.identifier.custom_callable, 770
angr.analyses.identifier.errors, 770
angr.analyses.identifier.func, 771
angr.analyses.identifier.functions, 771
angr.analyses.identifier.functions.atoi, 772
angr.analyses.identifier.functions.based_atoi, 772
angr.analyses.identifier.functions.fdprintf, 772
angr.analyses.identifier.functions.free, 773
angr.analyses.identifier.functions.int2str, 773
angr.analyses.identifier.functions.malloc, 774
angr.analyses.identifier.functions.memcmp, 775
angr.analyses.identifier.functions.memcpy, 775
angr.analyses.identifier.functions.memset, 776
angr.analyses.identifier.functions.printf, 776
angr.analyses.identifier.functions.recv_until, 777
angr.analyses.identifier.functions.skip_calloc, 778
angr.analyses.identifier.functions.skip_realloc, 778
angr.analyses.identifier.functions.skip_recv_n, 778
angr.analyses.identifier.functions.snprintf, 779
angr.analyses.identifier.functions.sprintf, 780
angr.analyses.identifier.functions.strcasecmp, 780
angr.analyses.identifier.functions.strcmp, 781
angr.analyses.identifier.functions.strcpy, 782
angr.analyses.identifier.functions.strlen, 782
angr.analyses.identifier.functions.strncmp, 783
angr.analyses.identifier.functions.strncpy, 784
angr.analyses.identifier.functions.strtol, 784
angr.analyses.identifier.identify, 784
angr.analyses.identifier.runner, 785
angr.analyses.init_finder, 786
angr.analyses.language_detector, 786
angr.analyses.loop_analysis, 787
angr.analyses.loop_analysis.loop_analysis, 787
angr.analyses.loop_unroller, 790
angr.analyses.loop_unroller.loop_unroller, 791
angr.analyses.loopfinder, 791
angr.analyses.outliner, 791
angr.analyses.outliner.outliner, 792
angr.analyses.patchfinder, 792
angr.analyses.pathfinder, 794
angr.analyses.propagator, 796
angr.analyses.propagator.engine_base, 796
angr.analyses.propagator.engine_vex, 797
angr.analyses.propagator.propagator, 797
angr.analyses.propagator.top_checker_mixin,

797
angr.analyses.propagator.values, 798
angr.analyses.propagator.vex_vars, 798
angr.analyses.proximity_graph, 798
angr.analyses.purity, 800
angr.analyses.purity.analysis, 801
angr.analyses.purity.engine, 801
angr.analyses.reaching_definitions, 804
angr.analyses.reaching_definitions.call_trace, 834
angr.analyses.reaching_definitions.dep_graph, 835
angr.analyses.reaching_definitions.engine_ail, 838
angr.analyses.reaching_definitions.engine_vex, 838
angr.analyses.reaching_definitions.external_codeloc, 838
angr.analyses.reaching_definitions.function_handler, 838
angr.analyses.reaching_definitions.function_handler_library, 845
angr.analyses.reaching_definitions.function_handler_variable, 846
angr.analyses.reaching_definitions.function_handler_variable_set, 850
angr.analyses.reaching_definitions.function_handler_variable_string, 852
angr.analyses.reaching_definitions.function_handler_variable_value_set, 855
angr.analyses.reaching_definitions.heap_allocator, 857
angr.analyses.reaching_definitions.rd_initializer, 857
angr.analyses.reaching_definitions.rd_state, 859
angr.analyses.reaching_definitions.reaching_definitions, 867
angr.analyses.reaching_definitions.subject, 869
angr.analyses.reassembler, 870
angr.analyses.s_liveness, 878
angr.analyses.s_propagator, 879
angr.analyses.s_reaching_definitions, 880
angr.analyses.s_reaching_definitions.s_rda_model, 885
angr.analyses.s_reaching_definitions.s_rda_view, 888
angr.analyses.s_reaching_definitions.s_reaching_definitions, 891
angr.analyses.smc, 891
angr.analyses.soot_class_hierarchy, 893
angr.analyses.stack_pointer_tracker, 893
angr.analyses.static_hooker, 896
angr.analyses.typehoon, 896
angr.analyses.typehoon.dfa, 898
angr.analyses.typehoon.simple_solver, 899
angr.analyses.typehoon.translator, 905
angr.analyses.typehoon.typeconsts, 906
angr.analyses.typehoon.typehoon, 913
angr.analyses.typehoon.typevars, 914
angr.analyses.typehoon.variance, 920
angr.analyses.unpacker, 921
angr.analyses.unpacker.obfuscation_detector, 921
angr.analyses.unpacker.packing_detector, 922
angr.analyses.variable_recovery, 922
angr.analyses.variable_recovery.annotations, 923
angr.analyses.variable_recovery.engine_ail, 923
angr.analyses.variable_recovery.engine_base, 924
angr.analyses.variable_recovery.engine_vex, 925
angr.analyses.variable_recovery.irsb_scanner, 925
angr.analyses.variable_recovery.variable_recovery, 925
angr.analyses.variable_recovery.variable_recovery_base, 926
angr.analyses.variable_recovery.variable_recovery_fast, 930
angr.analyses.veritesting, 930
angr.analyses.vfg, 932
angr.analyses.vsa_ddg, 936
angr.analyses.vtable, 937
angr.analyses.xrefs, 937
angr.angrdb, 938
angr.angrdb.db, 940
angr.angrdb.models, 942
angr.angrdb.serializers, 948
angr.angrdb.serializers.bookmarks, 949
angr.angrdb.serializers.callgraph, 950
angr.angrdb.serializers.cfg_model, 951
angr.angrdb.serializers.comments, 951
angr.angrdb.serializers.funcs, 951
angr.angrdb.serializers.kb, 952
angr.angrdb.serializers.labels, 952
angr.angrdb.serializers.loader, 953
angr.angrdb.serializers.structured_code, 954
angr.angrdb.serializers.variables, 955
angr.angrdb.serializers.xrefs, 956
angr.annocfg, 957
angr.blade, 958
angr.block, 959

angr.callable, 961
angr.calling_conventions, 962
angr.code_location, 985
angr.codenode, 987
angr.concretization_strategies, 988
angr.concretization_strategies.any, 990
angr.concretization_strategies.any_named, 990
angr.concretization_strategies.base, 990
angr.concretization_strategies.controlled_data, 991
angr.concretization_strategies.eval, 991
angr.concretization_strategies.logging, 991
angr.concretization_strategies.max, 991
angr.concretization_strategies.nonzero, 992
angr.concretization_strategies.nonzero_range, 992
angr.concretization_strategies.norepeats, 992
angr.concretization_strategies.norepeats_range, 992
angr.concretization_strategies.range, 992
angr.concretization_strategies.signed_add, 992
angr.concretization_strategies.single, 992
angr.concretization_strategies.solutions, 992
angr.concretization_strategies.unlimited_range, 993
angr.distributed, 993
angr.distributed.server, 994
angr.distributed.worker, 994
angr.emulator, 995
angr.engines, 996
angr.engines.ail, 1001
angr.engines.ail.callstack, 1003
angr.engines.ail.engine_light, 1003
angr.engines.ail.engine_successors, 1004
angr.engines.ail.setup, 1004
angr.engines.engine, 1004
angr.engines.failure, 1005
angr.engines.hook, 1005
angr.engines.icicle, 1005
angr.engines.light, 1005
angr.engines.light.data, 1009
angr.engines.light.engine, 1010
angr.engines.pcode, 1014
angr.engines.pcode.behavior, 1014
angr.engines.pcode.cc, 1034
angr.engines.pcode.emulate, 1036
angr.engines.pcode.engine, 1036
angr.engines.pcode.lifter, 1037
angr.engines.procedure, 1046
angr.engines.soot, 1046
angr.engines.soot.engine, 1047
angr.engines.soot.exceptions, 1047
angr.engines.soot.expressions, 1047
angr.engines.soot.expressions.arrayref, 1050
angr.engines.soot.expressions.base, 1050
angr.engines.soot.expressions.binop, 1050
angr.engines.soot.expressions.cast, 1050
angr.engines.soot.expressions.condition, 1050
angr.engines.soot.expressions.constants, 1050
angr.engines.soot.expressions.instancefieldref, 1051
angr.engines.soot.expressions.instanceOf, 1051
angr.engines.soot.expressions.invoke, 1051
angr.engines.soot.expressions.length, 1051
angr.engines.soot.expressions.local, 1051
angr.engines.soot.expressions.new, 1052
angr.engines.soot.expressions.newArray, 1052
angr.engines.soot.expressions.newMultiArray, 1052
angr.engines.soot.expressions.paramref, 1052
angr.engines.soot.expressions.phi, 1052
angr.engines.soot.expressions.staticfieldref, 1052
angr.engines.soot.expressions.thisref, 1052
angr.engines.soot.expressions.unsupported, 1052
angr.engines.soot.field_dispatcher, 1052
angr.engines.soot.method_dispatcher, 1053
angr.engines.soot.statements, 1053
angr.engines.soot.statements.assign, 1054
angr.engines.soot.statements.base, 1054
angr.engines.soot.statements.goto, 1054
angr.engines.soot.statements.identity, 1054
angr.engines.soot.statements.if_, 1054
angr.engines.soot.statements.invoke, 1054
angr.engines.soot.statements.return_, 1054
angr.engines.soot.statements.switch, 1055
angr.engines.soot.statements.throw, 1055
angr.engines.soot.values, 1055
angr.engines.soot.values.arrayref, 1059

angr.engines.soot.values.base, 1060
angr.engines.soot.values.constants, 1060
angr.engines.soot.values.instancefieldref, 1060
angr.engines.soot.values.local, 1060
angr.engines.soot.values.paramref, 1061
angr.engines.soot.values.staticfieldref, 1061
angr.engines.soot.values.strref, 1061
angr.engines.soot.values.thisref, 1062
angr.engines.successors, 1063
angr.engines.syscall, 1065
angr.engines.unicorn, 1065
angr.engines.vex, 1065
angr.engines.vex.claripy, 1068
angr.engines.vex.claripy.ccall, 1068
angr.engines.vex.claripy.datalayer, 1073
angr.engines.vex.claripy.irop, 1073
angr.engines.vex.heavy, 1074
angr.engines.vex.heavy.actions, 1075
angr.engines.vex.heavy.concretizers, 1075
angr.engines.vex.heavy.dirty, 1076
angr.engines.vex.heavy.heavy, 1078
angr.engines.vex.heavy.inspect, 1078
angr.engines.vex.heavy.resilience, 1079
angr.engines.vex.heavy.super_fastpath, 1079
angr.engines.vex.lifter, 1079
angr.engines.vex.light, 1080
angr.engines.vex.light.light, 1080
angr.engines.vex.light.resilience, 1080
angr.engines.vex.light.slicing, 1081
angr.errors, 1081
angr.exploration_techniques, 1087
angr.exploration_techniques.base, 1097
angr.exploration_techniques.bucketizer, 1098
angr.exploration_techniques.common, 1098
angr.exploration_techniques.dfs, 1099
angr.exploration_techniques.director, 1099
angr.exploration_techniques.driller_core, 1101
angr.exploration_techniques.explorer, 1102
angr.exploration_techniques.lengthlimiter, 1102
angr.exploration_techniques.local_loop_seer, 1102
angr.exploration_techniques.loop_seer, 1102
angr.exploration_techniques.manual_mergepoint, 1103
angr.exploration_techniques.memory_watcher, 1103
angr.exploration_techniques.oppologist, 1103
angr.exploration_techniques.slicecutor, 1104
angr.exploration_techniques.spiller, 1104
angr.exploration_techniques.spiller_db, 1105
angr.exploration_techniques.stochastic, 1106
angr.exploration_techniques.stub_stasher, 1106
angr.exploration_techniques.suggestions, 1106
angr.exploration_techniques.tech_builder, 1106
angr.exploration_techniques.threading, 1107
angr.exploration_techniques.timeout, 1107
angr.exploration_techniques.tracer, 1107
angr.exploration_techniques.unique, 1109
angr.exploration_techniques.veritesting, 1109
angr.factory, 1109
angr.flirt, 1113
angr.flirt.build_sig, 1114
angr.keyed_region, 1115
angr.knowledge_base, 1118
angr.knowledge_plugins, 1120
angr.knowledge_plugins.bookmarks, 1149
angr.knowledge_plugins.callsite_prototypes, 1150
angr.knowledge_plugins.cfg, 1152
angr.knowledge_plugins.cfg.block_id, 1165
angr.knowledge_plugins.cfg.cfg_manager, 1165
angr.knowledge_plugins.cfg.cfg_model, 1166
angr.knowledge_plugins.cfg.cfg_node, 1173
angr.knowledge_plugins.cfg.indirect_jump, 1175
angr.knowledge_plugins.cfg.memory_data, 1177
angr.knowledge_plugins.cfg.spilling_cfg, 1179
angr.knowledge_plugins.cfg.spilling_digraph, 1187
angr.knowledge_plugins.cfg.types, 1189
angr.knowledge_plugins.comments, 1189
angr.knowledge_plugins.custom_strings, 1191
angr.knowledge_plugins.data, 1191
angr.knowledge_plugins.debug_variables,

1192
angr.knowledge_plugins.functions, 1194
angr.knowledge_plugins.functions.function, 1209
angr.knowledge_plugins.functions.function_manager, 1218
angr.knowledge_plugins.functions.function_parser, 1227
angr.knowledge_plugins.functions.soot_function, 1228
angr.knowledge_plugins.indirect_jumps, 1228
angr.knowledge_plugins.key_definitions, 1228
angr.knowledge_plugins.key_definitions.atoms, 1241
angr.knowledge_plugins.key_definitions.constants, 1245
angr.knowledge_plugins.key_definitions.definition, 1245
angr.knowledge_plugins.key_definitions.environment, 1247
angr.knowledge_plugins.key_definitions.heap_address, 1248
angr.knowledge_plugins.key_definitions.key_definition, 1248
angr.knowledge_plugins.key_definitions.live_definitions, 1249
angr.knowledge_plugins.key_definitions.liveness, 1257
angr.knowledge_plugins.key_definitions.rd_model, 1259
angr.knowledge_plugins.key_definitions.tag, 1261
angr.knowledge_plugins.key_definitions.undefined, 1262
angr.knowledge_plugins.key_definitions.unknown_size, 1262
angr.knowledge_plugins.key_definitions.uses, 1262
angr.knowledge_plugins.labels, 1264
angr.knowledge_plugins.obfuscations, 1264
angr.knowledge_plugins.patches, 1264
angr.knowledge_plugins.plugin, 1266
angr.knowledge_plugins.propagations, 1266
angr.knowledge_plugins.propagations.prop_value, 1268
angr.knowledge_plugins.propagations.propagation_manager, 1270
angr.knowledge_plugins.propagations.propagation_model, 1271
angr.knowledge_plugins.propagations.states, 1272
angr.knowledge_plugins.rtdb, 1276
angr.knowledge_plugins.rtdb.rtdb, 1277
angr.knowledge_plugins.structured_code, 1278
angr.knowledge_plugins.types, 1280
angr.knowledge_plugins.variables, 1280
angr.knowledge_plugins.variables.spilling_vardict, 1282
angr.knowledge_plugins.variables.variable_access, 1282
angr.knowledge_plugins.variables.variable_manager, 1283
angr.knowledge_plugins.xrefs, 1290
angr.knowledge_plugins.xrefs.xref, 1291
angr.knowledge_plugins.xrefs.xref_manager, 1292
angr.knowledge_plugins.xrefs.xref_types, 1293
angr.llm_client, 1293
angr.llm_models, 1294
angr.mcp, 1295
angr.mcp.edit_tools, 1299
angr.mcp.errors, 1301
angr.mcp.serializers, 1301
angr.mcp.server, 1303
angr.mcp.session, 1307
angr.misc, 1309
angr.misc.ansi, 1314
angr.misc.autoimport, 1314
angr.misc.bug_report, 1315
angr.misc.hookset, 1315
angr.misc.loggers, 1316
angr.misc.picklable_lock, 1317
angr.misc.plugins, 1317
angr.misc.telemetry, 1320
angr.misc.testing, 1322
angr.misc.ux, 1322
angr.procedures, 1322
angr.procedures.advapi32, 1323
angr.procedures.cgc, 1323
angr.procedures.cgc.allocate, 1323
angr.procedures.cgc.deallocate, 1323
angr.procedures.cgc.fdwait, 1323
angr.procedures.cgc.random, 1323
angr.procedures.cgc.receive, 1324
angr.procedures.cgc.transmit, 1324
angr.procedures.definitions, 1324
angr.procedures.definitions.cgc, 1333
angr.procedures.definitions.gnulib, 1333
angr.procedures.definitions.libstdcpp, 1333
angr.procedures.definitions.linux_kernel, 1333
angr.procedures.definitions.linux_loader, 1333

angr.procedures.definitions.macho_libsystem, 1333
angr.procedures.definitions.msvcr, 1333
angr.procedures.definitions.parse_glibc, 1333
angr.procedures.definitions.parse_syscalls_from_local_system, 1334
angr.procedures.definitions.parse_win32json, 1334
angr.procedures.definitions.types_stl, 1334
angr.procedures.glibc, 1336
angr.procedures.glibc.dynamic_loading, 1336
angr.procedures.glibc.scanf, 1336
angr.procedures.glibc.sscanf, 1336
angr.procedures.gnulib, 1336
angr.procedures.gnulib.xalloc_die, 1337
angr.procedures.gnulib.xstrtol_fatal, 1337
angr.procedures.java, 1337
angr.procedures.java.unconstrained, 1337
angr.procedures.java_io, 1337
angr.procedures.java_io.read, 1337
angr.procedures.java_io.write, 1338
angr.procedures.java_jni, 1338
angr.procedures.java_jni.array_operations, 1338
angr.procedures.java_jni.class_and_interface_operations, 1340
angr.procedures.java_jni.field_access, 1340
angr.procedures.java_jni.global_and_local_refs, 1341
angr.procedures.java_jni.method_calls, 1342
angr.procedures.java_jni.not_implemented, 1346
angr.procedures.java_jni.object_operations, 1347
angr.procedures.java_jni.string_operations, 1347
angr.procedures.java_jni.version_information, 1347
angr.procedures.java_lang, 1348
angr.procedures.java_lang.character, 1348
angr.procedures.java_lang.double, 1348
angr.procedures.java_lang.exit, 1348
angr.procedures.java_lang.getsimplename, 1348
angr.procedures.java_lang.integer, 1348
angr.procedures.java_lang.load_library, 1349
angr.procedures.java_lang.math, 1349
angr.procedures.java_lang.string, 1349
angr.procedures.java_lang.stringbuilder, 1349
angr.procedures.java_lang.system, 1349
angr.procedures.java_util, 1349
angr.procedures.java_util.collection, 1350
angr.procedures.java_util.iterator, 1350
angr.procedures.java_util.list, 1350
angr.procedures.java_util.map, 1350
angr.procedures.java_util.random, 1350
angr.procedures.java_util.scanner_nextline, 1351
angr.procedures.libc, 1351
angr.procedures.libc.abort, 1353
angr.procedures.libc.access, 1353
angr.procedures.libc.atoi, 1353
angr.procedures.libc.atol, 1353
angr.procedures.libc.calloc, 1353
angr.procedures.libc.closelog, 1353
angr.procedures.libc.err, 1353
angr.procedures.libc.error, 1353
angr.procedures.libc.exit, 1353
angr.procedures.libc.fclose, 1354
angr.procedures.libc.feof, 1354
angr.procedures.libc.fflush, 1354
angr.procedures.libc.fgetc, 1354
angr.procedures.libc.fgets, 1354
angr.procedures.libc.fopen, 1354
angr.procedures.libc.fprintf, 1354
angr.procedures.libc.fputc, 1355
angr.procedures.libc.fputs, 1355
angr.procedures.libc.fread, 1355
angr.procedures.libc.free, 1355
angr.procedures.libc.fscanf, 1355
angr.procedures.libc.fseek, 1355
angr.procedures.libc.ftell, 1355
angr.procedures.libc.fwrite, 1356
angr.procedures.libc.getchar, 1356
angr.procedures.libc.getdelim, 1356
angr.procedures.libc.getegid, 1356
angr.procedures.libc.geteuid, 1356
angr.procedures.libc.getgid, 1356
angr.procedures.libc.gets, 1356
angr.procedures.libc.getuid, 1356
angr.procedures.libc.isalnum, 1356
angr.procedures.libc.isalpha, 1356
angr.procedures.libc.isascii, 1357
angr.procedures.libc.isblank, 1357
angr.procedures.libc.iscntrl, 1357
angr.procedures.libc.isdigit, 1357
angr.procedures.libc.isgraph, 1357
angr.procedures.libc.islower, 1357
angr.procedures.libc.isprint, 1357

angr.procedures.libc.ispunct, 1357
angr.procedures.libc.isspace, 1357
angr.procedures.libc.isupper, 1357
angr.procedures.libc.isxdigit, 1357
angr.procedures.libc.malloc, 1358
angr.procedures.libc.memccpy, 1358
angr.procedures.libc.memchr, 1358
angr.procedures.libc.memcmp, 1358
angr.procedures.libc.memcpy, 1358
angr.procedures.libc.memmove, 1358
angr.procedures.libc.mempcpy, 1358
angr.procedures.libc.memset, 1358
angr.procedures.libc.openlog, 1358
angr.procedures.libc.perror, 1358
angr.procedures.libc.printf, 1358
angr.procedures.libc.putchar, 1359
angr.procedures.libc.puts, 1359
angr.procedures.libc.rand, 1359
angr.procedures.libc.realloc, 1359
angr.procedures.libc.rewind, 1359
angr.procedures.libc.scanf, 1359
angr.procedures.libc.setbuf, 1359
angr.procedures.libc.setvbuf, 1359
angr.procedures.libc.snprintf, 1359
angr.procedures.libc.sprintf, 1359
angr.procedures.libc.srand, 1359
angr.procedures.libc.sscanf, 1360
angr.procedures.libc.stpcpy, 1360
angr.procedures.libc.strcat, 1360
angr.procedures.libc.strchr, 1360
angr.procedures.libc.strcmp, 1360
angr.procedures.libc.strcoll, 1360
angr.procedures.libc.strcpy, 1360
angr.procedures.libc.strlen, 1360
angr.procedures.libc.strncat, 1360
angr.procedures.libc.strncmp, 1360
angr.procedures.libc.strncpy, 1361
angr.procedures.libc.strnlen, 1361
angr.procedures.libc.strstr, 1361
angr.procedures.libc.strtok, 1361
angr.procedures.libc.strtol, 1361
angr.procedures.libc.strtoul, 1361
angr.procedures.libc.strxfrm, 1362
angr.procedures.libc.system, 1362
angr.procedures.libc.time, 1362
angr.procedures.libc.tmpnam, 1362
angr.procedures.libc.tolower, 1362
angr.procedures.libc.toupper, 1362
angr.procedures.libc.ungetc, 1362
angr.procedures.libc.vsnprintf, 1362
angr.procedures.libc.wchar, 1362
angr.procedures.libstdcpp, 1362
angr.procedures.libstdcpp.std____throw_bad_alloc, 1363
angr.procedures.libstdcpp.std____throw_bad_cast, 1363
angr.procedures.libstdcpp.std____throw_length_error, 1363
angr.procedures.libstdcpp.std____throw_logic_error, 1363
angr.procedures.libstdcpp.std__terminate, 1363
angr.procedures.linux_kernel, 1363
angr.procedures.linux_kernel.access, 1364
angr.procedures.linux_kernel.arch_prctl, 1364
angr.procedures.linux_kernel.arm_user_helpers, 1364
angr.procedures.linux_kernel.brk, 1364
angr.procedures.linux_kernel.cwd, 1365
angr.procedures.linux_kernel.fstat, 1365
angr.procedures.linux_kernel.fstat64, 1365
angr.procedures.linux_kernel.futex, 1365
angr.procedures.linux_kernel.getegid, 1365
angr.procedures.linux_kernel.geteuid, 1365
angr.procedures.linux_kernel.getgid, 1365
angr.procedures.linux_kernel.getpid, 1365
angr.procedures.linux_kernel.getrlimit, 1366
angr.procedures.linux_kernel.gettid, 1366
angr.procedures.linux_kernel.getuid, 1366
angr.procedures.linux_kernel.iovec, 1366
angr.procedures.linux_kernel.lseek, 1366
angr.procedures.linux_kernel.mmap, 1366
angr.procedures.linux_kernel.mprotect, 1366
angr.procedures.linux_kernel.munmap, 1366
angr.procedures.linux_kernel.openat, 1367
angr.procedures.linux_kernel.set_tid_address, 1367
angr.procedures.linux_kernel.sigaction, 1367
angr.procedures.linux_kernel.sigprocmask, 1367
angr.procedures.linux_kernel.stat, 1367
angr.procedures.linux_kernel.sysinfo, 1367
angr.procedures.linux_kernel.tgkill, 1367
angr.procedures.linux_kernel.time, 1367
angr.procedures.linux_kernel.uid, 1367
angr.procedures.linux_kernel.uname, 1368
angr.procedures.linux_kernel.unlink, 1368
angr.procedures.linux_kernel.vsyscall, 1368
angr.procedures.linux_loader, 1368

angr.procedures.linux_loader.sim_loader, 1368
angr.procedures.linux_loader.tls, 1368
angr.procedures.msvcr, 1369
angr.procedures.msvcr.fmode, 1369
angr.procedures.ntdll, 1369
angr.procedures.ntdll.exceptions, 1369
angr.procedures.posix, 1369
angr.procedures.posix.accept, 1370
angr.procedures.posix.bind, 1370
angr.procedures.posix.bzero, 1370
angr.procedures.posix.chroot, 1371
angr.procedures.posix.close, 1371
angr.procedures.posix.closedir, 1371
angr.procedures.posix.dup, 1371
angr.procedures.posix.fcntl, 1371
angr.procedures.posix.fdopen, 1371
angr.procedures.posix.fileno, 1371
angr.procedures.posix.fork, 1372
angr.procedures.posix.getenv, 1372
angr.procedures.posix.gethostbyname, 1372
angr.procedures.posix.getpass, 1372
angr.procedures.posix.getsockopt, 1372
angr.procedures.posix.htonl, 1372
angr.procedures.posix.htons, 1372
angr.procedures.posix.inet_ntoa, 1372
angr.procedures.posix.listen, 1373
angr.procedures.posix.mmap, 1373
angr.procedures.posix.open, 1373
angr.procedures.posix.opendir, 1373
angr.procedures.posix.poll, 1373
angr.procedures.posix.pread64, 1373
angr.procedures.posix.pthread, 1373
angr.procedures.posix.pwrite64, 1374
angr.procedures.posix.read, 1374
angr.procedures.posix.readdir, 1374
angr.procedures.posix.recv, 1374
angr.procedures.posix.recvfrom, 1374
angr.procedures.posix.select, 1374
angr.procedures.posix.send, 1374
angr.procedures.posix.setsockopt, 1375
angr.procedures.posix.sigaction, 1375
angr.procedures.posix.sim_time, 1375
angr.procedures.posix.sleep, 1375
angr.procedures.posix.socket, 1375
angr.procedures.posix.strcasecmp, 1375
angr.procedures.posix.strdup, 1375
angr.procedures.posix.strncasecmp, 1375
angr.procedures.posix.strndup, 1375
angr.procedures.posix.strtok_r, 1375
angr.procedures.posix.syslog, 1376
angr.procedures.posix.tz, 1376
angr.procedures.posix.unlink, 1376
angr.procedures.posix.usleep, 1376
angr.procedures.posix.write, 1376
angr.procedures.procedure_dict, 1376
angr.procedures.stubs, 1376
angr.procedures.stubs.b64_decode, 1378
angr.procedures.stubs.caller, 1378
angr.procedures.stubs.CallReturn, 1376
angr.procedures.stubs.crazy_scanf, 1378
angr.procedures.stubs.format_parser, 1378
angr.procedures.stubs.Nop, 1377
angr.procedures.stubs.NoReturnUnconstrained, 1377
angr.procedures.stubs.PathTerminator, 1377
angr.procedures.stubs.Redirect, 1377
angr.procedures.stubs.ReturnChar, 1377
angr.procedures.stubs.ReturnUnconstrained, 1377
angr.procedures.stubs.syscall_stub, 1380
angr.procedures.stubs.UnresolvableCallTarget, 1377
angr.procedures.stubs.UnresolvableJumpTarget, 1377
angr.procedures.stubs.UserHook, 1378
angr.procedures.testing, 1380
angr.procedures.testing.manyargs, 1380
angr.procedures.testing.retreg, 1380
angr.procedures.tracer, 1380
angr.procedures.tracer.random, 1381
angr.procedures.tracer.receive, 1381
angr.procedures.tracer.transmit, 1381
angr.procedures.uclibc, 1381
angr.procedures.win32, 1381
angr.procedures.win32.critical_section, 1383
angr.procedures.win32.dynamic_loading, 1383
angr.procedures.win32.EncodePointer, 1382
angr.procedures.win32.ExitProcess, 1382
angr.procedures.win32.file_handles, 1384
angr.procedures.win32.GetCommandLine, 1382
angr.procedures.win32.GetCurrentProcessId, 1382
angr.procedures.win32.GetCurrentThreadId, 1382
angr.procedures.win32.gethostbyname, 1384
angr.procedures.win32.GetLastInputInfo, 1382
angr.procedures.win32.GetModuleHandle, 1382
angr.procedures.win32.GetProcessAffinityMask, 1383
angr.procedures.win32.heap, 1384

angr.procedures.win32.InterlockedExchange, 1383
angr.procedures.win32.is_bad_ptr, 1384
angr.procedures.win32.IsProcessorFeaturePresent, 1383
angr.procedures.win32.local_storage, 1385
angr.procedures.win32.mutex, 1385
angr.procedures.win32.sim_time, 1386
angr.procedures.win32.system_paths, 1387
angr.procedures.win32.VirtualAlloc, 1383
angr.procedures.win32.VirtualProtect, 1383
angr.procedures.win32_kernel, 1387
angr.procedures.win32_kernel.ExAllocatePool, 1387
angr.procedures.win32_kernel.ExFreePoolWithTag, 1387
angr.procedures.win_user32, 1387
angr.procedures.win_user32.chars, 1387
angr.procedures.win_user32.keyboard, 1387
angr.procedures.win_user32.messagebox, 1388
angr.project, 1388
angr.protos, 1393
angr.protos.ail_types_pb2, 1393
angr.protos.cfg_pb2, 1395
angr.protos.clinic_pb2, 1396
angr.protos.codegen_pb2, 1397
angr.protos.decompilation_cache_pb2, 1402
angr.protos.function_pb2, 1402
angr.protos.primitives_pb2, 1403
angr.protos.variables_pb2, 1405
angr.protos.xrefs_pb2, 1406
angr.rust, 1406
angr.rust.analyses, 1406
angr.rust.analyses.cleanup_function_identification, 1406
angr.rust.analyses.flirt_sig_propagation, 1407
angr.rust.analyses.rust_calling_convention, 1407
angr.rust.analyses.rust_calling_convention.facade, 1408
angr.rust.analyses.rust_calling_convention.pathfinder, 1409
angr.rust.analyses.rust_calling_convention.rustcall, 1409
angr.rust.analyses.rust_calling_convention.rustcall_models, 1410
angr.rust.analyses.rust_symbol_recovery, 1410
angr.rust.analyses.rust_type_hints, 1410
angr.rust.analyses.rustc_version_identification, 1410
angr.rust.analyses.struct_builder, 1410
angr.rust.analyses.type_db_loader, 1411
angr.rust.knowledge_plugins, 1411
angr.rust.knowledge_plugins.clinic_factory, 1412
angr.rust.knowledge_plugins.known_structs, 1412
angr.rust.knowledge_plugins.librust, 1413
angr.rust.knowledge_plugins.rust_calling_conventions, 1413
angr.rust.knowledge_plugins.type_hints, 1413
angr.rust.mixins, 1413
angr.rust.mixins.cfa_mixin, 1417
angr.rust.mixins.cfg_transformation_mixin, 1418
angr.rust.mixins.dfa_mixin, 1418
angr.rust.mixins.srda_mixin, 1419
angr.rust.mixins.ssa_variable_mixin, 1420
angr.rust.optimization_passes, 1421
angr.rust.optimization_passes.cleanup_code_remover, 1421
angr.rust.optimization_passes.combo_register_rewriter, 1421
angr.rust.optimization_passes.deref_coercion_simplifie 1422
angr.rust.optimization_passes.deref_coercion_simplifie 1422
angr.rust.optimization_passes.error_propagation_simpli 1423
angr.rust.optimization_passes.function_prototype_infer 1423
angr.rust.optimization_passes.macro, 1423
angr.rust.optimization_passes.macro.format_macro_simpl 1423
angr.rust.optimization_passes.outliners, 1424
angr.rust.optimization_passes.outliners.string_cmp_out 1424
angr.rust.optimization_passes.outliners.string_literal 1425
angr.rust.optimization_passes.outliners.string_outline 1425
angr.rust.optimization_passes.outliners.unwrap_outline 1425
angr.rust.optimization_passes.outliners.vec_outliner, 1426
angr.rust.optimization_passes.pattern_match_simplifier 1426
angr.rust.optimization_passes.pre_pattern_match_simpli 1426
angr.rust.optimization_passes.redundant_block_remover, 1428
angr.rust.optimization_passes.ret_expr_rewriter,

1428
angr.rust.optimization_passes.rust_calling_convention, 1428
angr.rust.optimization_passes.security_check_removal, 1428
angr.rust.optimization_passes.str_argument_simplifier, 1429
angr.rust.optimization_passes.struct_instantiation_simplifier, 1429
angr.rust.optimization_passes.struct_return_simplifier, 1430
angr.rust.optimization_passes.utils, 1430
angr.rust.sim_type, 1431
angr.rust.structuring, 1435
angr.rust.structuring.structurer_nodes, 1436
angr.rust.typehoon, 1436
angr.rust.typehoon.translator, 1436
angr.rust.typehoon.typehoon, 1437
angr.rust.utils, 1437
angr.rust.utils.ail, 1437
angr.rust.utils.demangler, 1438
angr.rust.utils.rust_sigs, 1439
angr.rustylib, 1439
angr.serializable, 1440
angr.sim_manager, 1441
angr.sim_options, 1448
angr.sim_procedure, 1448
angr.sim_state, 1452
angr.sim_state_options, 1456
angr.sim_type, 1459
angr.sim_variable, 1477
angr.simos, 1479
angr.simos.cgc, 1486
angr.simos.javavm, 1486
angr.simos.linux, 1488
angr.simos.simos, 1488
angr.simos.snimmuc_nxp, 1491
angr.simos.uefi, 1491
angr.simos.userland, 1491
angr.simos.windows, 1492
angr.simos.xbox, 1493
angr.slicer, 1493
angr.state_hierarchy, 1494
angr.state_plugins, 1494
angr.state_plugins.callstack, 1542
angr.state_plugins.cgc, 1544
angr.state_plugins.debug_variables, 1545
angr.state_plugins.edge_hitmap, 1547
angr.state_plugins.filesystem, 1547
angr.state_plugins.gdb, 1551
angr.state_plugins.globals, 1551
angr.state_plugins.heap, 1552
angr.state_plugins.heap.heap_base, 1555
angr.state_plugins.heap.heap_brk, 1556
angr.state_plugins.heap.heap_freelist, 1557
angr.state_plugins.heap.heap_libc, 1558
angr.state_plugins.heap.heap_ptmalloc, 1559
angr.state_plugins.heap.utils, 1561
angr.state_plugins.history, 1561
angr.state_plugins.icicle, 1563
angr.state_plugins.inspect, 1565
angr.state_plugins.javavm_classloader, 1573
angr.state_plugins.jni_references, 1573
angr.state_plugins.libc, 1574
angr.state_plugins.light_registers, 1577
angr.state_plugins.log, 1577
angr.state_plugins.loop_data, 1577
angr.state_plugins.plugin, 1578
angr.state_plugins.posix, 1579
angr.state_plugins.preconstrainer, 1582
angr.state_plugins.scratch, 1583
angr.state_plugins.sim_action, 1584
angr.state_plugins.sim_action_object, 1585
angr.state_plugins.sim_event, 1587
angr.state_plugins.solver, 1587
angr.state_plugins.symbolizer, 1596
angr.state_plugins.trace_additions, 1596
angr.state_plugins.uc_manager, 1598
angr.state_plugins.unicorn_engine, 1599
angr.state_plugins.view, 1604
angr.storage, 1607
angr.storage.file, 1609
angr.storage.memory_mixins, 1617
angr.storage.memory_mixins.actions_mixin, 1633
angr.storage.memory_mixins.address_concretization_mixi 1633
angr.storage.memory_mixins.bvv_conversion_mixin, 1634
angr.storage.memory_mixins.conditional_store_mixin, 1634
angr.storage.memory_mixins.convenient_mappings_mixin, 1634
angr.storage.memory_mixins.default_filler_mixin, 1635
angr.storage.memory_mixins.dirty_addrs_mixin, 1635
angr.storage.memory_mixins.hex_dumper_mixin, 1635
angr.storage.memory_mixins.inspect_mixin, 1636
angr.storage.memory_mixins.javavm_memory_mixin, 1636

angr.storage.memory_mixins.keyvalue_memory_mixin, 1637
angr.storage.memory_mixins.label_merger_mixin, 1638
angr.storage.memory_mixins.memory_mixin, 1638
angr.storage.memory_mixins.multi_value_merger_mixin, 1641
angr.storage.memory_mixins.name_resolution_mixin, 1641
angr.storage.memory_mixins.paged_memory, 1641
angr.storage.memory_mixins.paged_memory.page_backers_mixin, 1641
angr.storage.memory_mixins.paged_memory.paged_memory_mixin, 1641
angr.storage.memory_mixins.paged_memory.paged_memory_value_mixin, 1643
angr.storage.memory_mixins.paged_memory.pages, 1643
angr.storage.memory_mixins.paged_memory.pages.base, 1647
angr.storage.memory_mixins.paged_memory.pages.cooperation, 1647
angr.storage.memory_mixins.paged_memory.pages.history_tracking_mixin, 1648
angr.storage.memory_mixins.paged_memory.pages.mixins, 1648
angr.storage.memory_mixins.paged_memory.pages.multi_page, 1649
angr.storage.memory_mixins.paged_memory.pages.utils, 1649
angr.storage.memory_mixins.paged_memory.pages.vectored_page, 1651
angr.storage.memory_mixins.paged_memory.pages.vectored_page_mixins, 1651
angr.storage.memory_mixins.paged_memory.pages.writeback_page, 1652
angr.storage.memory_mixins.paged_memory.pages.writeback_page_mixins, 1652
angr.storage.memory_mixins.paged_memory.privileged_mixin, 1654
angr.storage.memory_mixins.paged_memory.stack_allocation_mixin, 1655
angr.storage.memory_mixins.regioned_memory, 1655
angr.storage.memory_mixins.regioned_memory.abstract_address_descriptor, 1656
angr.storage.memory_mixins.regioned_memory.abstract_region_descriptor, 1659
angr.storage.memory_mixins.regioned_memory.region_data, 1659
angr.storage.memory_mixins.regioned_memory.region_meta, 1659
angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin, 1661
angr.storage.memory_mixins.regioned_memory.regioned_address_mixin, 1662
angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin, 1663
angr.storage.memory_mixins.regioned_memory.static_find, 1664
angr.storage.memory_mixins.simple_interface_mixin, 1664
angr.storage.memory_mixins.simplification_mixin, 1665
angr.storage.memory_mixins.size_resolution_mixin, 1665
angr.storage.memory_mixins.slotted_memory, 1665
angr.storage.memory_mixins.smart_find_mixin, 1666
angr.storage.memory_mixins.symbolic_merger_mixin, 1666
angr.storage.memory_mixins.top_merger_mixin, 1666
angr.storage.memory_mixins.underconstrained_mixin, 1666
angr.storage.memory_mixins.unwrapper_mixin, 1666
angr.storage.memory_object, 1666
angr.tablespecs, 1668
angr.utils, 1668
angr.utils.ail, 1669
angr.utils.ail_serialization, 1671
angr.utils.algo, 1673
angr.utils.balancer, 1673
angr.utils.bits, 1674
angr.utils.constants, 1675
angr.utils.cowdict, 1676
angr.utils.bitmap, 1677
angr.utils.doms, 1677
angr.utils.dynamic_dictlist, 1678
angr.utils.endness, 1678
angr.utils.enums_conv, 1679
angr.utils.env, 1679
angr.utils.formatting, 1679
angr.utils.funcid, 1680
angr.utils.go_runtime, 1680
angr.utils.graph, 1682
angr.utils.ins_addr_list, 1689
angr.utils.jsonify_utils, 1690
angr.utils.lazy_import, 1691
angr.utils.library, 1691
angr.utils.loader, 1693

angr.utils.mp, 1693
angr.utils.orderedset, 1695
angr.utils.smart_cache, 1695
angr.utils.ssa, 1695
angr.utils.ssa.combined_uses_collector, 1698
angr.utils.ssa.tmp_uses_collector, 1699
angr.utils.ssa.vvar_extra_defs_collector, 1699
angr.utils.ssa.vvar_uses_collector, 1699
angr.utils.strings, 1699
angr.utils.tagged_interval_map, 1700
angr.utils.timing, 1700
angr.utils.types, 1701
angr.utils.vex, 1703
angr.vaults, 1704
ModuloSimplifier (*class in angr.analyses.decompiler.peephole_optimizations.modulo_simplifier*), 613
most_mergeable() (*angr.state_hierarchy.StateHierarchy method*), 1494
mount() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
move() (*angr.sim_manager.SimulationManager method*), 1445
move_codelocs() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 863
mp_context() (*in module angr.utils.mp*), 1694
mprotect (*class in angr.procedures.linux_kernel.mprotect*), 1366
Mul (*angr.engines.light.ArithmeticExpression attribute*), 1006
Mul (*angr.engines.light.data.ArithmeticExpression attribute*), 1010
mulpyplex() (*angr.sim_manager.SimulationManager method*), 1442
MultiNode (*class in angr.analyses.decompiler.structurer_nodes*), 710
MultipleBlocksException, 586
MultiStatementExpression (*class in angr.ailment.expression*), 221
MultiStatementExpressionAssignmentFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 635
MultiStmtExprMode (*class in angr.analyses.decompiler.structuring.phoenix*), 717
MultiValuedMemory (*class in angr.storage.memory_mixins*), 1625
MultiValueMergerMixin (*class in angr.storage.memory_mixins.multi_value_merger_mixin*), 1641
MultiValues (*class in angr.storage.memory_mixins.paged_memory.pages.multi_values*), 1649
MultiwriteAnnotation (*class in angr.storage.memory_mixins.address_concretization_mixin*), 1633
munmap (*class in angr.procedures.linux_kernel.munmap*), 1366
mutate_dict() (*in module angr.procedures.win32.local_storage*), 1385
mv_is_bv() (*in module angr.storage.memory_mixins.paged_memory.pages.multi_values*), 1650
mv_is_fp() (*in module angr.storage.memory_mixins.paged_memory.pages.multi_values*), 1650
MVListPage (*class in angr.storage.memory_mixins.paged_memory.pages.m*), 1651
MVListPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mix*), 1643
MVListPagesWithLabelsMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mix*), 1643

## N

name (*angr.ailment.expression.ConvertType attribute*), 219
name (*angr.ailment.expression.VirtualVariableCategory attribute*), 223
name (*angr.ailment.ExpressionKind attribute*), 212
name (*angr.analyses.decompiler.edits.resolve.ResolvedVariable property*), 447
name (*angr.analyses.decompiler.known_patterns.dsl.PAny attribute*), 492
name (*angr.analyses.decompiler.known_patterns.dsl.PBinOp attribute*), 494
name (*angr.analyses.decompiler.known_patterns.dsl.PCall attribute*), 497
name (*angr.analyses.decompiler.known_patterns.dsl.PCallResult attribute*), 498
name (*angr.analyses.decompiler.known_patterns.dsl.PConst attribute*), 493
name (*angr.analyses.decompiler.known_patterns.dsl.PConv attribute*), 495
name (*angr.analyses.decompiler.known_patterns.dsl.PDefOf attribute*), 498
name (*angr.analyses.decompiler.known_patterns.dsl.PExtract attribute*), 496
name (*angr.analyses.decompiler.known_patterns.dsl.PField attribute*), 499
name (*angr.analyses.decompiler.known_patterns.dsl.PITE attribute*), 501
name (*angr.analyses.decompiler.known_patterns.dsl.PLoad attribute*), 496

name (*angr.analyses.decompiler.known_patterns.dsl.PPhi attribute*), 501
name (*angr.analyses.decompiler.known_patterns.dsl.PStackVar attribute*), 500
name (*angr.analyses.decompiler.known_patterns.dsl.PUnaryOp attribute*), 494
name (*angr.analyses.decompiler.known_patterns.dsl.PVVar attribute*), 492
name (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 463
name (*angr.analyses.decompiler.known_patterns.PAny attribute*), 464
name (*angr.analyses.decompiler.known_patterns.PBinOp attribute*), 465
name (*angr.analyses.decompiler.known_patterns.PCall attribute*), 466
name (*angr.analyses.decompiler.known_patterns.PCallResult attribute*), 467
name (*angr.analyses.decompiler.known_patterns.PConst attribute*), 469
name (*angr.analyses.decompiler.known_patterns.PConv attribute*), 470
name (*angr.analyses.decompiler.known_patterns.PDefOf attribute*), 470
name (*angr.analyses.decompiler.known_patterns.PExtract attribute*), 471
name (*angr.analyses.decompiler.known_patterns.PField attribute*), 472
name (*angr.analyses.decompiler.known_patterns.PITE attribute*), 453
name (*angr.analyses.decompiler.known_patterns.PLoad attribute*), 473
name (*angr.analyses.decompiler.known_patterns.PPhi attribute*), 474
name (*angr.analyses.decompiler.known_patterns.PStackField attribute*), 475
name (*angr.analyses.decompiler.known_patterns.PUnaryOp attribute*), 476
name (*angr.analyses.decompiler.known_patterns.PVVar attribute*), 477
name (*angr.analyses.decompiler.known_patterns.templates.PatternTemplate attribute*), 532
NAME (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePtrSaveSimplifier attribute*), 560
NAME (*angr.analyses.decompiler.optimization_passes.BasePtrSaveSimplifier attribute*), 540
NAME (*angr.analyses.decompiler.optimization_passes.call_stmt_rewriter.CallStatementRewriter attribute*), 560
NAME (*angr.analyses.decompiler.optimization_passes.CallStatementRewriter attribute*), 540
NAME (*angr.analyses.decompiler.optimization_passes.code_motion.CodeMotionOptimization attribute*), 561
NAME (*angr.analyses.decompiler.optimization_passes.CodeMotionOptimization attribute*), 541
NAME (*angr.analyses.decompiler.optimization_passes.condition_constprop.C attribute*), 562
NAME (*angr.analyses.decompiler.optimization_passes.ConditionConstantPro attribute*), 542
NAME (*angr.analyses.decompiler.optimization_passes.const_derefs.Constant attribute*), 563
NAME (*angr.analyses.decompiler.optimization_passes.const_prop_reverter.C attribute*), 563
NAME (*angr.analyses.decompiler.optimization_passes.ConstantDereferences attribute*), 543
NAME (*angr.analyses.decompiler.optimization_passes.ConstPropOptReverte attribute*), 542
NAME (*angr.analyses.decompiler.optimization_passes.cross_jump_reverter.C attribute*), 564
NAME (*angr.analyses.decompiler.optimization_passes.CrossJumpReverter attribute*), 543
NAME (*angr.analyses.decompiler.optimization_passes.deadblock_remover.De attribute*), 565
NAME (*angr.analyses.decompiler.optimization_passes.DeadblockRemover attribute*), 544
NAME (*angr.analyses.decompiler.optimization_passes.determine_load_sizes. attribute*), 565
NAME (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSimp attribute*), 566
NAME (*angr.analyses.decompiler.optimization_passes.DivSimplifier attribute*), 544
NAME (*angr.analyses.decompiler.optimization_passes.duplication_reverter.d attribute*), 570
NAME (*angr.analyses.decompiler.optimization_passes.duplication_reverter.D attribute*), 566
NAME (*angr.analyses.decompiler.optimization_passes.DuplicationReverter attribute*), 544
NAME (*angr.analyses.decompiler.optimization_passes.eager_std_string_conc attribute*), 574
NAME (*angr.analyses.decompiler.optimization_passes.eager_std_string_eval attribute*), 574
NAME (*angr.analyses.decompiler.optimization_passes.EagerStdStringConcat attribute*), 546
NAME (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.Exp attribute*), 576
NAME (*angr.analyses.decompiler.optimization_passes.ExprOpSwapper attribute*), 547
NAME (*angr.analyses.decompiler.optimization_passes.flip_boolean_cmp.Flip attribute*), 576
NAME (*angr.analyses.decompiler.optimization_passes.FlipBooleanCmp attribute*), 547
NAME (*angr.analyses.decompiler.optimization_passes.inlined_memcpy_simp attribute*), 577
NAME (*angr.analyses.decompiler.optimization_passes.inlined_memcpy_simp attribute*), 577
NAME (*angr.analyses.decompiler.optimization_passes.inlined_memset_simpl attribute*), 578
NAME (*angr.analyses.decompiler.optimization_passes.inlined_memset_simpl attribute*), 578

NAME (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 578
NAME (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 579
NAME (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 581
NAME (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 581
NAME (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 582
NAME (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 582
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 548
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 548
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 548
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 548
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 549
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 549
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 549
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 550
NAME (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 550
NAME (*angr.analyses.decompiler.optimization_passes.ite_expr_converters.ITEExpressionConverter attribute*), 583
NAME (*angr.analyses.decompiler.optimization_passes.ite_region_converter.ITERegionConverter attribute*), 583
NAME (*angr.analyses.decompiler.optimization_passes.ITEExpressionConverter attribute*), 547
NAME (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 548
NAME (*angr.analyses.decompiler.optimization_passes.known_argument_call_simplifier.KnownArgumentCallSimplifier attribute*), 583
NAME (*angr.analyses.decompiler.optimization_passes.KnownArgumentCallSimplifier attribute*), 550
NAME (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 585
NAME (*angr.analyses.decompiler.optimization_passes.LoweredSwitchSimplifier attribute*), 551
NAME (*angr.analyses.decompiler.optimization_passes.mips_gp_setting_optimizer.MipsGpSettingOptimizer attribute*), 585
NAME (*angr.analyses.decompiler.optimization_passes.MipsGpSettingOptimizer attribute*), 552
NAME (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 586
NAME (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 552
NAME (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifier attribute*), 587
NAME (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifierLate attribute*), 590
NAME (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 590
NAME (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifierLate attribute*), 591
NAME (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifier attribute*), 553
NAME (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 591
NAME (*angr.analyses.decompiler.optimization_passes.ret_deduplicator.ReturnDeduplicator attribute*), 591
NAME (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSimplifier attribute*), 553
NAME (*angr.analyses.decompiler.optimization_passes.return_duplicator_high.ReturnDuplicatorHigh attribute*), 593
NAME (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.ReturnDuplicatorLow attribute*), 593
NAME (*angr.analyses.decompiler.optimization_passes.ReturnDeduplicator attribute*), 554
NAME (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHigh attribute*), 554
NAME (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorLow attribute*), 555
NAME (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 594
NAME (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifier attribute*), 556
NAME (*angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.StaticVVarRewriter attribute*), 596
NAME (*angr.analyses.decompiler.optimization_passes.StaticVVarRewriter attribute*), 556
NAME (*angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator.SwitchDefaultCaseDuplicator attribute*), 596
NAME (*angr.analyses.decompiler.optimization_passes.switch_reused_entry_to_other_case.SwitchReusedEntryRewriter attribute*), 597
NAME (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator attribute*), 557
NAME (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryRewriter attribute*), 557
NAME (*angr.analyses.decompiler.optimization_passes.tag_slicer.TagSlicer attribute*), 597
NAME (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 558
NAME (*angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier.WinStackCanarySimplifier attribute*), 598
NAME (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier attribute*), 558

NAME (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier* attribute), 598
NAME (*angr.analyses.decompiler.optimization_passes.X86GccGetPcSimplifier* attribute), 558
NAME (*angr.analyses.decompiler.peephole_optimizations.a_div_const_add_a_mul_n_div_const_shr.ConvAddAMulNDivConstShr* attribute), 600
NAME (*angr.analyses.decompiler.peephole_optimizations.a_mul_const_div_shr_const_mul_const_div_shr_const* attribute), 600
NAME (*angr.analyses.decompiler.peephole_optimizations.a_mul_const_sub_a_mul_const_sub_a* attribute), 600
NAME (*angr.analyses.decompiler.peephole_optimizations.a_sub_const_sub_a_sub_const_sub_a* attribute), 601
NAME (*angr.analyses.decompiler.peephole_optimizations.a_sub_a_div_sub_a_div* attribute), 601
NAME (*angr.analyses.decompiler.peephole_optimizations.a_sub_a_sub_a_sub_a_sub_a* attribute), 601
NAME (*angr.analyses.decompiler.peephole_optimizations.a_sub_a_sub_a_sub_a_sub_a* attribute), 601
NAME (*angr.analyses.decompiler.peephole_optimizations.arm_ifarm_ifarm_ifarm_ifarm* attribute), 602
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationExpressionBase* attribute), 604
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationMultiStmtBase* attribute), 603
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationStmtBase* attribute), 602
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationOffsetAddN* attribute), 605
NAME (*angr.analyses.decompiler.peephole_optimizations.base.PeepholeOptimizationOffsetAndMask* attribute), 605
NAME (*angr.analyses.decompiler.peephole_optimizations.bitwise_ifarm_ifarm_ifarm_ifarm* attribute), 605
NAME (*angr.analyses.decompiler.peephole_optimizations.bitwise_ifarm_ifarm_ifarm_ifarm* attribute), 606
NAME (*angr.analyses.decompiler.peephole_optimizations.bool_ifarm_ifarm_ifarm_ifarm* attribute), 606
NAME (*angr.analyses.decompiler.peephole_optimizations.bswap_swap* attribute), 606
NAME (*angr.analyses.decompiler.peephole_optimizations.bt_ifarm_ifarm_ifarm_ifarm_ifarm* attribute), 606
NAME (*angr.analyses.decompiler.peephole_optimizations.cas_ifarm_ifarm_ifarm_ifarm* attribute), 607
NAME (*angr.analyses.decompiler.peephole_optimizations.cmp_ifarm_ifarm_ifarm_ifarm* attribute), 608
NAME (*angr.analyses.decompiler.peephole_optimizations.cmp_ifarm_ifarm_ifarm_ifarm* attribute), 608
NAME (*angr.analyses.decompiler.peephole_optimizations.cmp_ifarm_ifarm_ifarm_ifarm* attribute), 609
NAME (*angr.analyses.decompiler.peephole_optimizations.coalescer_ifarm_ifarm_ifarm_ifarm* attribute), 609
NAME (*angr.analyses.decompiler.peephole_optimizations.coalescer_ifarm_ifarm_ifarm_ifarm* attribute), 609
NAME (*angr.analyses.decompiler.peephole_optimizations.constant_derefs.ConstantDerefs.Coalescer* attribute), 610
NAME (*angr.analyses.decompiler.peephole_optimizations.constant_derefs.ConstantDerefs.Coalescer* attribute), 610
NAME (*angr.analyses.decompiler.peephole_optimizations.conv_a_sub0_shr_* attribute), 610
NAME (*angr.analyses.decompiler.peephole_optimizations.conv_a_sub0_shr_* attribute), 610
NAME (*angr.analyses.decompiler.peephole_optimizations.conv_a_sub0_shr_* attribute), 610
NAME (*angr.analyses.decompiler.peephole_optimizations.eager_eval.EagerE* attribute), 611
NAME (*angr.analyses.decompiler.peephole_optimizations.extended_byte_and* attribute), 612
NAME (*angr.analyses.decompiler.peephole_optimizations.invert_negated_log* attribute), 612
NAME (*angr.analyses.decompiler.peephole_optimizations.known_pattern_ca* attribute), 612
NAME (*angr.analyses.decompiler.peephole_optimizations.modulo_simplifier.* attribute), 613
NAME (*angr.analyses.decompiler.peephole_optimizations.one_sub_bool.One* attribute), 613
NAME (*angr.analyses.decompiler.peephole_optimizations.optimized_div_sim* attribute), 613
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_cascading_* attribute), 614
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_const_inse* attribute), 614
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_cxx_destru* attribute), 614
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_empty_if_b* attribute), 614
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_noop_conv* attribute), 615
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 615
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 615
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 616
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 616
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 616
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 617
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 617
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 617
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 617
NAME (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_* attribute), 618
NAME (*angr.analyses.decompiler.peephole_optimizations.rewrite_bit_extract* attribute), 618
NAME (*angr.analyses.decompiler.peephole_optimizations.rewrite_conv_mul.* attribute), 618

NAME (*angr.analyses.decompiler.peephole_optimizations.rewrite_cxx_operator_calls.RewriteCXXOperatorCalls attribute*), 619
NAME (*angr.analyses.decompiler.peephole_optimizations.rewrite_mips_gp_loads.RewriteMipsGpLoads attribute*), 619
NAME (*angr.analyses.decompiler.peephole_optimizations.rol_ror.RolRorRewriter attribute*), 619
NAME (*angr.analyses.decompiler.peephole_optimizations.sar_to_signed_div.SarToSignedDiv attribute*), 620
NAME (*angr.analyses.decompiler.peephole_optimizations.shl_to_multiply.ShlToMultiply attribute*), 620
NAME (*angr.analyses.decompiler.peephole_optimizations.simplify_ctz_clz_patterns.SimplifyCtzClzPatterns attribute*), 620
NAME (*angr.analyses.decompiler.peephole_optimizations.single_bit_cond_to_boolexpr.SingleBitCondToBoolExpr attribute*), 620
NAME (*angr.analyses.decompiler.peephole_optimizations.single_bit_xor.SingleBitXor attribute*), 621
NAME (*angr.analyses.decompiler.peephole_optimizations.tidy_stack_addr.TidyStackAddr attribute*), 621
name (*angr.analyses.decompiler.structured_codegen.c.CVariable property*), 679
NAME (*angr.analyses.decompiler.structuring.dream.DreamStructurer attribute*), 716
NAME (*angr.analyses.decompiler.structuring.DreamStructurer attribute*), 714
NAME (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer attribute*), 717
NAME (*angr.analyses.decompiler.structuring.PhoenixStructurer attribute*), 714
NAME (*angr.analyses.decompiler.structuring.sailr.SAILRStructurer attribute*), 719
NAME (*angr.analyses.decompiler.structuring.SAILRStructurer attribute*), 716
NAME (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase attribute*), 719
NAME (*angr.analyses.deobfuscator.api_obf_peephole_optimizer.APIObfPeepholeOptimizer attribute*), 735
NAME (*angr.analyses.deobfuscator.api_obf_peephole_optimizer_base.APIObfPeepholeOptimizerBase attribute*), 735
NAME (*angr.analyses.deobfuscator.APIObfType1PeepholeOptimizer attribute*), 731
NAME (*angr.analyses.deobfuscator.APIObfType3PeepholeOptimizer attribute*), 732
NAME (*angr.analyses.deobfuscator.string_obf_opt_passes.StringObfRewriterBase attribute*), 739
NAME (*angr.analyses.deobfuscator.string_obf_peephole_optimizer_base.StringObfPeepholeOptimizerBase attribute*), 739
NAME (*angr.analyses.deobfuscator.StringObfType1PeepholeOptimizer attribute*), 733
NAME (*angr.analyses.deobfuscator.StringObfType3Rewriter attribute*), 733
name (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
name (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
name (*angr.analyses.reaching_definitions.function_handler.FunctionHandler property*), 873
name (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
name (*angr.knowledge_plugins.functions.function.Function property*), 1211
name (*angr.knowledge_plugins.key_definitions.atoms.Register property*), 1244
name (*angr.procedures.definitions.SimLibrary property*), 1325
NAME (*angr.rust.optimization_passes.simplify_reference_loads.SimplifyReferenceLoads attribute*), 1421
NAME (*angr.rust.optimization_passes.single_bit_cond_to_bool_expr.SingleBitCondToBoolExpr attribute*), 1422
NAME (*angr.rust.optimization_passes.single_bit_xor.SingleBitXor attribute*), 1422
NAME (*angr.rust.optimization_passes.tidy_stack_addr.TidyStackAddr attribute*), 1422
NAME (*angr.rust.optimization_passes.error_propagation_simplifier.ErrorPropagationSimplifier attribute*), 1423
NAME (*angr.rust.optimization_passes.function_prototype_inference.FunctionPrototypeInference attribute*), 1423
NAME (*angr.rust.optimization_passes.macro.format_macro_simplifier.FormatMacroSimplifier attribute*), 1424
NAME (*angr.rust.optimization_passes.outliners.string_cmp_outliner.StringCmpOutliner attribute*), 1424
NAME (*angr.rust.optimization_passes.outliners.string_literal_outliner.StringLiteralOutliner attribute*), 1425
NAME (*angr.rust.optimization_passes.outliners.string_outliner.StringOutliner attribute*), 1425
NAME (*angr.rust.optimization_passes.outliners.unwrap_outliner.UnwrapOutliner attribute*), 1426
NAME (*angr.rust.optimization_passes.outliners.vec_outliner.VecOutliner attribute*), 1426
NAME (*angr.rust.optimization_passes.pattern_match_simplifier.PatternMatchSimplifier attribute*), 1426
NAME (*angr.rust.optimization_passes.pre_pattern_match_simplifier.PrePatternMatchSimplifier attribute*), 1427
NAME (*angr.rust.optimization_passes.redundant_block_remover.RedundantBlockRemover attribute*), 1428
NAME (*angr.rust.optimization_passes.ret_expr_rewriter.RetExprRewriter attribute*), 1428
NAME (*angr.rust.optimization_passes.rust_calling_convention.RustCallingConvention attribute*), 1428
NAME (*angr.rust.optimization_passes.security_check_remover.SecurityCheckRemover attribute*), 1429
NAME (*angr.rust.optimization_passes.str_argument_simplifier.StrArgumentSimplifier attribute*), 1429
NAME (*angr.rust.optimization_passes.struct_instantiation_simplifier.StructInstantiationSimplifier attribute*), 1430
NAME (*angr.rust.optimization_passes.struct_return_simplifier.StructReturnSimplifier attribute*), 1430
name (*angr.sim_type.NamedTypeMixin property*), 1462
name (*angr.sim_type.SimTypeRef property*), 1475

name (*angr.sim_type.TypeRef property*), 1461
name_stop() (*angr.state_plugins.unicorn_engine.STOP static method*), 1602
name_to_pass() (*in module angr.analyses.decompiler.optimization_pass_registry*), 540
NameCollisionError, 437
NamedTypeMixin (*class in angr.sim_type*), 1461
NameResolutionMixin (*class in angr.storage.memory_mixins.name_resolution_mixin*), 1641
native_arch (*angr.simos.javavm.SimJavaVM property*), 1487
nbits (*angr.utils.tagged_interval_map.TaggedIntervalMap property*), 1700
nbytes (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap.SymbolicBitmap property*), 1652
NE (*angr.analyses.decompiler.region_simplifiers.switch_clusters.CmpOp attribute*), 639
needs_details (*angr.knowledge_plugins.propagations.prop_value.PropValue property*), 1269
negate() (*in module angr.ailment.expression*), 223
NEVER (*angr.analyses.decompiler.structuring.phoenix.MultiStatusExpressionRef attribute*), 717
new (*angr.analyses.decompiler.edits.EditResult attribute*), 424
new (*angr.analyses.decompiler.edits.results.EditResult attribute*), 450
new() (*angr.analyses.cfg.cfg_job_base.FunctionKey static method*), 326
new() (*angr.analyses.typehoon.typeconsts.Pointer method*), 910
new() (*angr.knowledge_plugins.cfg.block_id.BlockID static method*), 1165
new_array() (*angr.engines.soot.expressions.newArray.SimSootExpr_NewArray static method*), 1052
new_array() (*angr.engines.soot.expressions.newMultiArray.SimSootExpr_NewMultiArray static method*), 1052
new_block_addr() (*angr.analyses.decompiler.clinic.Clinic method*), 407
new_block_addr() (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass method*), 588
new_dtv() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 917
new_dtv_with_merged_labels() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 917
new_from_shared() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage class method*), 1654
new_label() (*angr.analyses.reassembler.Label static method*), 870
new_label() (*angr.analyses.reassembler.SymbolManager method*), 871
new_model() (*angr.knowledge_plugins.cfg.cfg_manager.CFGManager method*), 1165
new_object() (*angr.engines.soot.values.thisref.SimSootValue_ThisRef class method*), 1062
new_stack_vvar() (*angr.rust.mixins.ssa_variable_mixin.SSAVariableMix method*), 1420
new_string() (*angr.engines.soot.values.strref.SimSootValue_StringRef static method*), 1061
new_tv() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 916
NewArray (*class in angr.procedures.java_jni.array_operations*), 1338
NewBooleanArray (*class in angr.procedures.java_jni.array_operations*), 1338
NewByteArray (*class in angr.procedures.java_jni.array_operations*), 1338
NewCharArray (*class in angr.procedures.java_jni.array_operations*), 1338
NewFunctionHandler (*class in angr.analyses.find_objects_static*), 746
NewGlobalRef (*class in angr.procedures.java_jni.global_and_local_refs*), 1341
NewIntArray (*class in angr.procedures.java_jni.array_operations*), 1339
NewLocalRef (*class in angr.procedures.java_jni.global_and_local_refs*), 1341
NewLongArray (*class in angr.procedures.java_jni.array_operations*), 1339
NewObject (*class in angr.procedures.java_jni.object_operations*), 1347
NewShortArray (*class in angr.procedures.java_jni.array_operations*), 1339
NewStringUTF (*class in angr.procedures.java_jni.string_operations*), 1347
next_arg() (*angr.calling_conventions.SimCC method*), 969
next_arg() (*angr.calling_conventions.SimCCARM method*), 977
next_arg() (*angr.calling_conventions.SimCCARMHF method*), 977
next_arg() (*angr.calling_conventions.SimCCCdecl method*), 972
next_arg() (*angr.calling_conventions.SimCCGoAMD64

*method*), 975
next_arg() (*angr.calling_conventions.SimCCMicrosoftAMD64 method*), 973
next_arg() (*angr.calling_conventions.SimCCMicrosoftFastcall method*), 972
next_arg() (*angr.calling_conventions.SimCCO32 method*), 980
next_arg() (*angr.calling_conventions.SimCCRISCV64 method*), 979
next_arg() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 975
next_arg() (*angr.calling_conventions.SimCCUsercall method*), 971
next_atom() (*angr.ailment.Manager method*), 212
next_atom() (*angr.ailment.manager.Manager method*), 224
next_bookmark() (*angr.knowledge_plugins.bookmarks.Bookmarks method*), 1150
next_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
next_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 1560
next_clear() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap.SymbolicBitmap method*), 1653
next_free_pos() (*angr.rustylib.SegmentList method*), 1439
next_id() (*angr.analyses.typehoon.typevars.TypeVariableManager method*), 916
next_ident() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 670
next_idx() (*angr.analyses.decompiler.known_patterns.dsl.MatchCtx method*), 491
next_node() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 767
next_node_idx() (*angr.analyses.decompiler.optimization_passes.return_duplicator_base.ReturnDuplicatorBase method*), 592
next_node_idx() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 670
next_pos_with_sort_not_in() (*angr.rustylib.SegmentList method*), 1439
next_set() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap.SymbolicBitmap method*), 1653
next_variable_ident() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1283
NextInt (*class in angr.procedures.java_util.random*), 1350
nilref() (*in module angr.analyses.pathfinder*), 794
NO_MAINBIN_BACKENDS (*angr.angrdb.serializers.loader.LoaderSerializer attribute*), 953
NO_MAINBIN_BACKENDS (*angr.angrdb.serializers.LoaderSerializer attribute*), 948
no_ret (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
NO_RET (*angr.procedures.gnulib.xalloc_die.xalloc_die attribute*), 1337
NO_RET (*angr.procedures.gnulib.xstrtol_fatal.xstrtol_fatal attribute*), 1337
NO_RET (*angr.procedures.java_lang.exit.Exit attribute*), 1348
NO_RET (*angr.procedures.libc.abort.abort attribute*), 1353
NO_RET (*angr.procedures.libc.err.err attribute*), 1353
NO_RET (*angr.procedures.libc.error.error attribute*), 1353
NO_RET (*angr.procedures.libc.exit.exit attribute*), 1353
NO_RET (*angr.procedures.libstdcpp.std____throw_bad_alloc.std____throw_ attribute*), 1363
NO_RET (*angr.procedures.libstdcpp.std____throw_bad_cast.std____throw_b attribute*), 1363
NO_RET (*angr.procedures.libstdcpp.std____throw_length_error.std____thro attribute*), 1363
NO_RET (*angr.procedures.libstdcpp.std____throw_logic_error.std____throw attribute*), 1363
NO_RET (*angr.procedures.libstdcpp.std__terminate.std__terminate attribute*), 1363
NO_RET (*angr.procedures.linux_loader.sim_loader.IFuncResolver attribute*), 1368
NO_RET (*angr.procedures.linux_loader.sim_loader.LinuxLoader attribute*), 1368
NO_RET (*angr.procedures.stubs.CallReturn.CallReturn attribute*), 1376
NO_RET (*angr.procedures.stubs.NoReturnUnconstrained.NoReturnUnconstr attribute*), 1377
NO_RET (*angr.procedures.stubs.PathTerminator.PathTerminator attribute*), 1377
NO_RET (*angr.procedures.stubs.Redirect.Redirect attribute*), 1377
NO_RET (*angr.procedures.stubs.UnresolvableCallTarget.UnresolvableCallTa attribute*), 1377
NO_RET (*angr.procedures.stubs.UnresolvableJumpTarget.UnresolvableJump attribute*), 1377
NO_RET (*angr.procedures.stubs.UserHook.UserHook attribute*), 1378
NO_RET (*angr.procedures.testing.manyargs.manyargs attribute*), 1380
NO_RET (*angr.procedures.win32.ExitProcess.ExitProcess attribute*), 1382
NO_RET (*angr.sim_procedure.SimProcedure attribute*), 1450
NO_RET (*angr.SimProcedure attribute*), 190
NO_RM (*angr.engines.vex.claripy.irop.SimIROp attribute*), 1074
NO_SUCCESSORS (*angr.emulator.EmulatorStopReason attribute*), 996
NO_SUCCESSORS (*angr.EmulatorStopReason attribute*), 161

NoConcreteDispatch, 893
NodalAnnotation (*class in angr.analyses.data_dep.data_dependency_analysis*), 358
node_addr_has_return() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1167
node_addrs (*angr.knowledge_plugins.cfg.cfg_model.CFGModel property*), 1166
node_config (*angr.analyses.decompiler.structured_codegen.c_serializable.SerializableCConstruct property*), 689
node_keys (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1182
node_observe() (*angr.analyses.reaching_definitions.reaching_definitions_analysis.ReachingDefinitionsAnalysis method*), 868
node_position() (*angr.analyses.decompiler.structurer_nodes.SequenceNode method*), 711
NodeAddressFinder (*class in angr.analyses.decompiler.region_simplifiers.node_address_finder*), 638
NoDecode (*angr.protos.primitives_pb2.Edge attribute*), 1403
NodeConfigEntry (*class in angr.protos.codegen_pb2*), 1401
NodeConfigMsg (*class in angr.protos.codegen_pb2*), 1401
NodeFoundNotification, 582
NodeReplacer (*class in angr.analyses.decompiler.node_replacer*), 535
nodes (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 349
nodes (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1183
nodes (*angr.knowledge_plugins.functions.function.Function property*), 1212
nodes() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 767
nodes() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 835
nodes() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1169
nodes_by_addr() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1166
nodes_by_addr() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1183
non_null (*angr.analyses.identifier.functions.fdprintf.fdprintf attribute*), 772
non_null (*angr.analyses.identifier.functions.memcpy.memcpy attribute*), 775
non_null (*angr.analyses.identifier.functions.memset.memset attribute*), 776
non_null (*angr.analyses.identifier.functions.printf.printf attribute*), 776
non_null (*angr.analyses.identifier.functions.snprintf.snprintf attribute*), 779
non_null (*angr.analyses.identifier.functions.sprintf.sprintf attribute*), 780
non_null (*angr.analyses.identifier.functions.strcasecmp.strcasecmp attribute*), 780
non_null (*angr.analyses.identifier.functions.strcmp.strcmp attribute*), 781
non_null (*angr.analyses.identifier.functions.strcpy.strcpy attribute*), 782
non_null (*angr.analyses.identifier.functions.strlen.strlen attribute*), 782
non_null (*angr.analyses.identifier.functions.strncmp.strncmp attribute*), 783
non_null (*angr.analyses.identifier.functions.strncpy.strncpy attribute*), 784
non_zero_exprs() (*angr.knowledge_plugins.propagations.prop_value.PropValue method*), 1269
NONE (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 1209
NONE (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
NONE (*angr.sim_type.PointerDisposition attribute*), 1466
NONE (*angr.simos.windows.SecurityCookieInit attribute*), 1492
nonreturning_func_addrs() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1224
NoOp (*class in angr.ailment.statement*), 225
Nop (*class in angr.procedures.stubs.Nop*), 1377
NORedir (*angr.protos.primitives_pb2.Edge attribute*), 1403
NoReturnUnconstrained (*class in angr.procedures.stubs.NoReturnUnconstrained*), 1377
NORMAL (*angr.analyses.cfg.cfg_fast.CFGJobType attribute*), 319
normalize() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
normalize() (*angr.knowledge_plugins.functions.function.Function method*), 1217
normalize() (*angr.knowledge_plugins.key_definitions.definition.Definition method*), 1246
normalize() (*angr.rust.sim_type.RustSimTypeFunction method*), 1432
normalize() (*in module angr.rust.utils.demangler*), 1438
normalize_cpp_function_name() (*in module angr.sim_type*), 1477
normalize_go_func_name() (*in module angr.utils.go_runtime*), 1682
normalized (*angr.analyses.cfg.cfg_base.CFGBase property*), 308
NormalizedBlock (*class in angr.analyses.bindiff*), 280

NormalizedFunction (*class in angr.analyses.bindiff*), 280
NotAJumpTableNotification, 333
NotDecompiledError, 437
NotMemoryview (*class in angr.storage.memory_mixins.paged_memory.page_backer_mixins*), 1641
NotypeLabel (*class in angr.analyses.reassembler*), 871
NULL_TERMINATE (*angr.knowledge_plugins.key_definitions.num_fixes attribute*), 1229
NULL_TERMINATE (*angr.knowledge_plugins.key_definitions.num_defs attribute*), 1249
NullEditHooks (*class in angr.analyses.decompiler.edits.hooks*), 440
num_args() (*angr.analyses.identifier.func.Func method*), 771
num_args() (*angr.analyses.identifier.functions.atoi.atoi method*), 772
num_args() (*angr.analyses.identifier.functions.based_atoi.based_atoi method*), 772
num_args() (*angr.analyses.identifier.functions.fdprintf.fdprintf method*), 773
num_args() (*angr.analyses.identifier.functions.free.free method*), 773
num_args() (*angr.analyses.identifier.functions.int2str.int2str method*), 774
num_args() (*angr.analyses.identifier.functions.int2str.int2str_v2 method*), 774
num_args() (*angr.analyses.identifier.functions.int2str.int2str_v3 method*), 774
num_args() (*angr.analyses.identifier.functions.int2str.int2str_v4 method*), 774
num_args() (*angr.analyses.identifier.functions.malloc.malloc method*), 774
num_args() (*angr.analyses.identifier.functions.memcmp.memcmp method*), 775
num_args() (*angr.analyses.identifier.functions.memcpy.memcpy method*), 775
num_args() (*angr.analyses.identifier.functions.memset.memset method*), 776
num_args() (*angr.analyses.identifier.functions.printf.printf method*), 777
num_args() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 778
num_args() (*angr.analyses.identifier.functions.recv_until.receive_until_fd method*), 777
num_args() (*angr.analyses.identifier.functions.skip_calloc.calloc method*), 778
num_args() (*angr.analyses.identifier.functions.skip_realloc.realloc method*), 778
num_args() (*angr.analyses.identifier.functions.skip_recv_n.receive_n2 method*), 779
num_args() (*angr.analyses.identifier.functions.skip_recv_n.receive_n3 method*), 779
num_args() (*angr.analyses.identifier.functions.skip_recv_n.receive_n4 method*), 778
num_args() (*angr.analyses.identifier.functions.snprintf.snprintf method*), 779
num_args() (*angr.analyses.identifier.functions.sprintf.sprintf method*), 780
num_args() (*angr.analyses.identifier.functions.strcasecmp.strcasecmp method*), 781
num_args() (*angr.analyses.identifier.functions.strcmp.strcmp method*), 781
num_args() (*angr.analyses.identifier.functions.strcpy.strcpy method*), 782
num_args() (*angr.analyses.identifier.functions.strlen.strlen method*), 783
num_args() (*angr.analyses.identifier.functions.strncmp.strncmp method*), 783
num_args() (*angr.analyses.identifier.functions.strncpy.strncpy method*), 784
num_args() (*angr.analyses.identifier.functions.strtol.strtol method*), 784
num_arguments (*angr.knowledge_plugins.functions.function.Function property*), 1213
num_predecessors() (*angr.rust.mixins.cfa_mixin.CFAMixin method*), 1417
num_successors() (*angr.rust.mixins.cfa_mixin.CFAMixin method*), 1417
num_variants() (*angr.rust.sim_type.RustSimEnum method*), 1435
number_of_edges() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCF method*), 1185
number_of_nodes() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCF method*), 1183

## O

O (*in module angr.analyses.decompiler.decompilation_options*), 414
O_ACCMODE (*angr.storage.file.Flags attribute*), 1609
O_APPEND (*angr.storage.file.Flags attribute*), 1609
O_ASYNC (*angr.storage.file.Flags attribute*), 1609
O_CLOEXEC (*angr.storage.file.Flags attribute*), 1609
O_CREAT (*angr.storage.file.Flags attribute*), 1609
O_DIRECT (*angr.storage.file.Flags attribute*), 1609
O_DIRECTORY (*angr.storage.file.Flags attribute*), 1609
O_DSYNC (*angr.storage.file.Flags attribute*), 1609
O_EXCL (*angr.storage.file.Flags attribute*), 1609
O_LARGEFILE (*angr.storage.file.Flags attribute*), 1609
O_NDELAY (*angr.storage.file.Flags attribute*), 1610
O_NOATIME (*angr.storage.file.Flags attribute*), 1610
O_NOCTTY (*angr.storage.file.Flags attribute*), 1610
O_NOFOLLOW (*angr.storage.file.Flags attribute*), 1610
O_NONBLOCK (*angr.storage.file.Flags attribute*), 1610
O_PATH (*angr.storage.file.Flags attribute*), 1610
O_RDONLY (*angr.storage.file.Flags attribute*), 1609
O_RDWR (*angr.storage.file.Flags attribute*), 1609

O_SYNC (*angr.storage.file.Flags attribute*), 1610
O_TMPFILE (*angr.storage.file.Flags attribute*), 1610
O_TRUNC (*angr.storage.file.Flags attribute*), 1610
O_WRONLY (*angr.storage.file.Flags attribute*), 1609
ObfuscationDetector (*class in angr.analyses.unpacker.obfuscation_detector*), 921
Obfuscations (*class in angr.knowledge_plugins.obfuscations*), 1264
obj (*angr.procedures.java_jni.object_operations.NewObject attribute*), 1347
obj (*angr.utils.graph.ContainerNode property*), 1685
obj_bit_size() (*in module angr.storage.memory_object*), 1666
obj_id (*angr.keyed_region.StoredObject property*), 1115
ObjectLabel (*class in angr.analyses.reassembler*), 871
ObservationPointType (*class in angr.knowledge_plugins.key_definitions.constants*), 1245
observe() (*angr.analyses.s_reaching_definitions.s_rda_view.SRDAView method*), 891
observed_results (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 868
occupied_by() (*angr.rustylib.SegmentList method*), 1439
occupied_by_sort() (*angr.rustylib.SegmentList method*), 1439
occupied_size (*angr.rustylib.SegmentList attribute*), 1439
occupy() (*angr.rustylib.SegmentList method*), 1439
odd_offsets_are_zero() (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier static method*), 582
offset (*angr.analyses.decompiler.known_patterns.dsl.PExtract attribute*), 496
offset (*angr.analyses.decompiler.known_patterns.PExtract attribute*), 471
offset (*angr.analyses.reassembler.Label property*), 870
offset (*angr.analyses.stack_pointer_tracker.OffsetVal property*), 894
offset (*angr.knowledge_plugins.functions.function.Function property*), 1213
offset (*angr.knowledge_plugins.key_definitions.definition.Definition property*), 1247
Offset (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 1293
Offset (*angr.knowledge_plugins.xrefs.XRefType attribute*), 1291
Offset (*class in angr.analyses.decompiler.optimization_passes.duplication_reverter*), 595
offset_after() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
offset_after_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
offset_before() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
offset_before_block() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
offset_of() (*angr.analyses.decompiler.known_patterns.generator.PatternGenerator method*), 516
offset_sort_key() (*in module angr.analyses.decompiler.ssailification.traversal_engine*), 655
offsets (*angr.sim_type.SimStruct property*), 1471
offsets_for() (*angr.analyses.stack_pointer_tracker.StackPointerTracker method*), 896
OffsetVal (*class in angr.analyses.stack_pointer_tracker*), 894
offsIP (*angr.engines.pcode.lifter.IRSB property*), 1040
OK (*angr.misc.telemetry.StatusCode attribute*), 1321
old (*angr.analyses.decompiler.edits.EditResult attribute*), 124
old (*angr.analyses.decompiler.edits.results.EditResult attribute*), 750
old_mmap (*class in angr.procedures.linux_kernel.mmap*), 1366
on_message() (*angr.mcp.server.StdoutGuardMiddleware method*), 1303
on_worker_exit() (*angr.distributed.server.Server method*), 994
once() (*in module angr.misc.ux*), 1322
one_defat (*angr.knowledge_plugins.propagations.prop_value.PropValue property*), 1269
one_expr (*angr.knowledge_plugins.propagations.prop_value.PropValue property*), 1269
one_label() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 918
one_result (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 868
one_type() (*angr.sim_state_options.StateOption method*), 1456
one_value() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 1649
OneSubBool (*class in angr.analyses.decompiler.peephole_optimizations.one_sub_bool*), 613
OneTwoOrThree (*class in angr.analyses.identifier.functions.based_atoi*), 772
op (*angr.state_plugins.sim_action_object.SimActionObject property*), 1586
Op (*class in angr.ailment.expression*), 221
OP_AFTER (*angr.analyses.reaching_definitions.ObservationPointType attribute*), 820
OP_AFTER (*angr.knowledge_plugins.key_definitions.constants.ObservationPointType attribute*), 1245
op_attrs() (*in module angr.engines.vex.claripy.irop*),

1073
OP_BEFORE (*angr.analyses.reaching_definitions.ObservationPointType attribute*), 820
OP_BEFORE (*angr.knowledge_plugins.key_definitions.constants.ObservationPointType attribute*), 1245
op_concretize() (*in module angr.engines.vex.claripy.ccall*), 1068
op_precedence (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp property*), 681
op_precedence (*angr.analyses.decompiler.structured_codegen.rust.RustBinaryOp property*), 703
op_str (*angr.block.CapstoneInsn property*), 960
op_str (*angr.block.DisassemblerInsn property*), 959
op_str (*angr.block.PCodeInsn property*), 960
op_str (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 1037
OPAQUE_ELEMENT_SIZES (*in module angr.procedures.definitions.types_stl*), 1335
OpBehavior (*class in angr.engines.pcode.behavior*), 1014
OpBehaviorBoolAnd (*class in angr.engines.pcode.behavior*), 1028
OpBehaviorBoolNegate (*class in angr.engines.pcode.behavior*), 1027
OpBehaviorBoolOr (*class in angr.engines.pcode.behavior*), 1028
OpBehaviorBoolXor (*class in angr.engines.pcode.behavior*), 1027
OpBehaviorCopy (*class in angr.engines.pcode.behavior*), 1015
OpBehaviorEqual (*class in angr.engines.pcode.behavior*), 1016
OpBehaviorFloatAbs (*class in angr.engines.pcode.behavior*), 1031
OpBehaviorFloatAdd (*class in angr.engines.pcode.behavior*), 1030
OpBehaviorFloatCeil (*class in angr.engines.pcode.behavior*), 1032
OpBehaviorFloatDiv (*class in angr.engines.pcode.behavior*), 1030
OpBehaviorFloatEqual (*class in angr.engines.pcode.behavior*), 1029
OpBehaviorFloatFloat2Float (*class in angr.engines.pcode.behavior*), 1031
OpBehaviorFloatFloor (*class in angr.engines.pcode.behavior*), 1032
OpBehaviorFloatInt2Float (*class in angr.engines.pcode.behavior*), 1031
OpBehaviorFloatLess (*class in angr.engines.pcode.behavior*), 1029
OpBehaviorFloatLessEqual (*class in angr.engines.pcode.behavior*), 1029
OpBehaviorFloatMult (*class in angr.engines.pcode.behavior*), 1030
OpBehaviorFloatNan (*class in angr.engines.pcode.behavior*), 1029
OpBehaviorFloatNeg (*class in angr.engines.pcode.behavior*), 1031
OpBehaviorFloatNotEqual (*class in angr.engines.pcode.behavior*), 1029
OpBehaviorFloatRound (*class in angr.engines.pcode.behavior*), 1032
OpBehaviorFloatSqrt (*class in angr.engines.pcode.behavior*), 1031
OpBehaviorFloatSub (*class in angr.engines.pcode.behavior*), 1030
OpBehaviorFloatTrunc (*class in angr.engines.pcode.behavior*), 1032
OpBehaviorInt2Comp (*class in angr.engines.pcode.behavior*), 1021
OpBehaviorIntAdd (*class in angr.engines.pcode.behavior*), 1019
OpBehaviorIntAnd (*class in angr.engines.pcode.behavior*), 1023
OpBehaviorIntCarry (*class in angr.engines.pcode.behavior*), 1020
OpBehaviorIntDiv (*class in angr.engines.pcode.behavior*), 1025
OpBehaviorIntLeft (*class in angr.engines.pcode.behavior*), 1023
OpBehaviorIntLess (*class in angr.engines.pcode.behavior*), 1018
OpBehaviorIntLessEqual (*class in angr.engines.pcode.behavior*), 1018
OpBehaviorIntMult (*class in angr.engines.pcode.behavior*), 1025
OpBehaviorIntNegate (*class in angr.engines.pcode.behavior*), 1022
OpBehaviorIntOr (*class in angr.engines.pcode.behavior*), 1023
OpBehaviorIntRem (*class in angr.engines.pcode.behavior*), 1026
OpBehaviorIntRight (*class in angr.engines.pcode.behavior*), 1024
OpBehaviorIntSborrow (*class in angr.engines.pcode.behavior*), 1021
OpBehaviorIntScarry (*class in angr.engines.pcode.behavior*), 1021
OpBehaviorIntSdiv (*class in angr.engines.pcode.behavior*), 1026
OpBehaviorIntSext (*class in angr.engines.pcode.behavior*), 1019
OpBehaviorIntSless (*class in angr.engines.pcode.behavior*), 1017
OpBehaviorIntSlessEqual (*class in angr.engines.pcode.behavior*), 1017
OpBehaviorIntSrem (*class in angr.engines.pcode.behavior*), 1026

OpBehaviorIntSright (*class in angr.engines.pcode.behavior*), 1024
OpBehaviorIntSub (*class in angr.engines.pcode.behavior*), 1020
OpBehaviorIntXor (*class in angr.engines.pcode.behavior*), 1022
OpBehaviorIntZext (*class in angr.engines.pcode.behavior*), 1018
OpBehaviorLzcount (*class in angr.engines.pcode.behavior*), 1033
OpBehaviorNotEqual (*class in angr.engines.pcode.behavior*), 1016
OpBehaviorPiece (*class in angr.engines.pcode.behavior*), 1032
OpBehaviorPopcount (*class in angr.engines.pcode.behavior*), 1033
OpBehaviorSubpiece (*class in angr.engines.pcode.behavior*), 1033
Opcode (*class in angr.analyses.disassembly*), 741
OpDescriptor (*class in angr.analyses.decompiler.optimization_passes.expr_op_swapper*), 575
open (*class in angr.procedures.posix.open*), 1373
open() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
open_db() (*angr.angrdb.db.AngrDB method*), 940
open_db() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method*), 1278
open_socket() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
openat (*class in angr.procedures.linux_kernel.openat*), 1367
opendir (*class in angr.procedures.posix.opendir*), 1373
openlog (*class in angr.procedures.libc.openlog*), 1358
opens() (*angr.analyses.decompiler.known_patterns.gating.AllOf method*), 513
opens() (*angr.analyses.decompiler.known_patterns.gating.AnyOf method*), 513
opens() (*angr.analyses.decompiler.known_patterns.gating.CorroboratedBy method*), 512
opens() (*angr.analyses.decompiler.known_patterns.gating.CorroboratedByPattern method*), 514
opens() (*angr.analyses.decompiler.known_patterns.gating.PatternGate method*), 511
opens() (*angr.analyses.decompiler.known_patterns.gating.TargetGate method*), 512
Operand (*class in angr.analyses.disassembly*), 741
Operand (*class in angr.analyses.reassembler*), 871
operand_str (*angr.analyses.reassembler.DataLabel property*), 870
operand_str (*angr.analyses.reassembler.FunctionLabel property*), 871
operand_str (*angr.analyses.reassembler.Label property*), 870
operand_str (*angr.analyses.reassembler.NotypeLabel property*), 871
operand_str (*angr.analyses.reassembler.ObjectLabel property*), 871
OperandPiece (*class in angr.analyses.disassembly*), 742
OPERATE (*angr.state_plugins.sim_action.SimActionData attribute*), 1585
OPERATE (*angr.state_plugins.SimActionData attribute*), 1504
operations (*angr.engines.pcode.lifter.IRSB property*), 1040
operations (*angr.knowledge_plugins.functions.function.Function property*), 1213
OPERATOR_DELETE (*in module angr.analyses.decompiler.known_patterns.std_string_dtor*), 526
OperatorCounter (*class in angr.analyses.decompiler.counters.expression_counters*), 412
Oppologist (*class in angr.exploration_techniques.oppologist*), 1103
OPT_LEVELS (*angr.rust.analyses.rust_symbol_recovery.RustSymbolRecover attribute*), 1410
OPT_LEVELS (*angr.rust.analyses.rustc_version_identification.RustcVersionI attribute*), 1410
OptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 587
OptimizationPassStage (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 586
optimize() (*angr.analyses.binary_optimizer.BinaryOptimizer method*), 280
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_div_cons method*), 600
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_mul_con method*), 600
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_mul_con method*), 600
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_shl_cons method*), 601
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_sub_a_d method*), 601
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_sub_a_s method*), 601
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_sub_a_s method*), 601
optimize() (*angr.analyses.decompiler.peephole_optimizations.arm_cmpf method*), 602
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 604
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph method*), 603
optimize() (*angr.analyses.decompiler.peephole_optimizations.base.Peeph*

*method*), 603
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_sub_const_and_cmp_const_to_bool_const.ASubConstAndCmpConstToBool* *method*), 605
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_sub_const_and_cmp_const_to_bool_const.ASubConstAndCmpZeroToBool* *method*), 605
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_xor_a_const_to_const.AXorAConstToConst* *method*), 605
optimize() (*angr.analyses.decompiler.peephole_optimizations.a_xor_b_xor_a_to_b.AXorBXorAToB* *method*), 606
optimize() (*angr.analyses.decompiler.peephole_optimizations.bail_addition_to_boolean_expression.BailAdditionToBooleanExpression* *method*), 606
optimize() (*angr.analyses.decompiler.peephole_optimizations.bail_addition_to_boolean_or.BailAdditionToBooleanOr* *method*), 606
optimize() (*angr.analyses.decompiler.peephole_optimizations.bail_addition_to_multiplication.BailAdditionToMultiplication* *method*), 606
optimize() (*angr.analyses.decompiler.peephole_optimizations.bail_addition_to_sub.BailAdditionToSub* *method*), 607
optimize() (*angr.analyses.decompiler.peephole_optimizations.bail_shift_to_multiplication_signed.BailShiftToMultiplicationSigned* *method*), 608
optimize() (*angr.analyses.decompiler.peephole_optimizations.bail_shift_to_multiplication_unsigned.BailShiftToMultiplicationUnsigned* *method*), 608
optimize() (*angr.analyses.decompiler.peephole_optimizations.bit_extraction.BitExtraction* *method*), 609
optimize() (*angr.analyses.decompiler.peephole_optimizations.bitwise_and_to_logical_and.BitwiseAndToLogicalAnd* *method*), 609
optimize() (*angr.analyses.decompiler.peephole_optimizations.bitwise_or_to_logical_or.BitwiseOrToLogicalOr* *method*), 609
optimize() (*angr.analyses.decompiler.peephole_optimizations.bool_to_int.BoolToInt* *method*), 610
optimize() (*angr.analyses.decompiler.peephole_optimizations.cascading_comparisons.CascadingComparisons* *method*), 610
optimize() (*angr.analyses.decompiler.peephole_optimizations.cmp_ord_to_cmp_ord.CmpORDToCmpORD* *method*), 610
optimize() (*angr.analyses.decompiler.peephole_optimizations.conv_cmp_to_bool.ConvCmpToBool* *method*), 611
optimize() (*angr.analyses.decompiler.peephole_optimizations.double_negation.DoubleNegation* *method*), 611
optimize() (*angr.analyses.decompiler.peephole_optimizations.eager_evaluation.EagerEvaluation* *method*), 611
optimize() (*angr.analyses.decompiler.peephole_optimizations.invert_negated_logical_conjunction_disjunction.InvertNegatedLogicalConjunctionsAndDisjunctions* *method*), 612
optimize() (*angr.analyses.decompiler.peephole_optimizations.mod_simplifier.ModSimplifier* *method*), 612
optimize() (*angr.analyses.decompiler.peephole_optimizations.mul_neg.MulNeg* *method*), 613
optimize() (*angr.analyses.decompiler.peephole_optimizations.mull_simplifier.MullSimplifier* *method*), 613
optimize() (*angr.analyses.decompiler.peephole_optimizer.PeepholeOptimizerStmt* *method*), 613
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_cast_around_const_load.RemoveCastAroundConstLoad* *method*), 614
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_cast_around_load.RemoveCastAroundLoad* *method*), 614
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_empty_ite.RemoveEmptyIfThenElse* *method*), 614
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_empty_ite.RemoveEmptyIfThenElseMatch* *method*), 615
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_noop_conversions.RemoveNoopConversions* *method*), 615
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasks* *method*), 615
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 615
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 616
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 616
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 616
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 617
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 617
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 617
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 617
optimize() (*angr.analyses.decompiler.peephole_optimizations.remove_redundant_bitmasks.RemoveRedundantBitmasksExtended* *method*), 618
optimize() (*angr.analyses.decompiler.peephole_optimizations.rewrite_bit_extract_to_bit_cmp.RewriteBitExtractToBitCmp* *method*), 618
optimize() (*angr.analyses.decompiler.peephole_optimizations.rewrite_conv_to_bool.RewriteConvToBool* *method*), 618
optimize() (*angr.analyses.decompiler.peephole_optimizations.rewrite_cxx_exception_eh_ptr.RewriteCxxExceptionEhPtr* *method*), 619
optimize() (*angr.analyses.decompiler.peephole_optimizations.rewrite_mips_gp_load.RewriteMipsGpLoad* *method*), 619
optimize() (*angr.analyses.decompiler.peephole_optimizations.rol_ror.RolRor* *method*), 619
optimize() (*angr.analyses.decompiler.peephole_optimizations.sar_to_signmask.SarToSignmask* *method*), 620
optimize() (*angr.analyses.decompiler.peephole_optimizations.shl_to_multiplication.ShlToMultiplication* *method*), 620
optimize() (*angr.analyses.decompiler.peephole_optimizations.simplify_comparisons_against_zero.SimplifyComparisonsAgainstZero* *method*), 620
optimize() (*angr.analyses.decompiler.peephole_optimizations.single_bit_cmp.SingleBitCmp* *method*), 621
optimize() (*angr.analyses.decompiler.peephole_optimizations.single_bit_cmp.SingleBitCmpStmt* *method*), 621
optimize() (*angr.analyses.decompiler.peephole_optimizations.tidy_stack_addr.TidyStackAddr* *method*), 621
optimize() (*angr.analyses.deobfuscation.cff_simplifier_peephole_optimizer.APIOptimizer* *method*), 735
optimize() (*angr.analyses.deobfuscation.string_obf_peephole_optimizer.APIOptimizer* *method*), 736
optimize() (*angr.analyses.deobfuscation.string_obf_peephole_optimizer.StrOptimizer*

*method*), 739
optimize_binaryop() (*angr.analyses.decompiler.peephole_optimizations.optimized_divisions.OptimizedDivisionSimplifier method*), 613
OptimizedDivisionSimplifier (*class in angr.analyses.decompiler.peephole_optimizations.optimized_divisions*), 613
OptionalInt64 (*class in angr.protos.cfg_pb2*), 1396
OptionEntry (*class in angr.protos.decompilation_cache_pb2*), 1402
OPTIONS (*angr.analyses.cfg.cfg_arch_options.CFGArchOptions attribute*), 307
OPTIONS (*angr.analyses.cfg.CFGArchOptions attribute*), 292
OPTIONS (*angr.analyses.CFGArchOptions attribute*), 237
OPTIONS (*angr.sim_state_options.SimStateOptions attribute*), 1456
options_to_params() (*angr.analyses.decompiler.decompiler.Decompiler static method*), 417
Or (*angr.engines.light.ArithmeticExpression attribute*), 1006
Or (*angr.engines.light.data.ArithmeticExpression attribute*), 1009
Or1 (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransformationTypes attribute*), 334
ordered (*angr.analyses.decompiler.known_patterns.dsl.PStmtSeq attribute*), 505
ordered (*angr.analyses.decompiler.known_patterns.PStmtSeq attribute*), 475
OrderedSet (*class in angr.utils.orderedset*), 1695
original_size (*angr.ailment.Block attribute*), 210
original_size (*angr.ailment.block.Block attribute*), 215
OS_MSDOS (*angr.analyses.flirt.consts.FlirtOSType attribute*), 754
OS_NETWARE (*angr.analyses.flirt.consts.FlirtOSType attribute*), 754
OS_OS2 (*angr.analyses.flirt.consts.FlirtOSType attribute*), 754
OS_OTHER (*angr.analyses.flirt.consts.FlirtOSType attribute*), 754
OS_UNIX (*angr.analyses.flirt.consts.FlirtOSType attribute*), 754
OS_WIN (*angr.analyses.flirt.consts.FlirtOSType attribute*), 754
other_types (*angr.procedures.stubs.format_parser.FormatParser attribute*), 1379
others (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
OUT (*angr.sim_type.PointerDisposition attribute*), 1466
out_degree (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1185
out_degree_by_key() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1186
out_edges (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property*), 1185
out_edges_by_key() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1185
OutdatedError, 306
OuterWalker (*class in angr.analyses.decompiler.optimization_passes.expr_op_swapper*), 575
outline() (*angr.analyses.decompiler.known_patterns.finder.KnownPatternFinder method*), 510
outline_at() (*angr.analyses.decompiler.known_patterns.finder.KnownPatternFinder method*), 510
Outliner (*class in angr.analyses.outliner.outliner*), 792
OutlineResult (*class in angr.analyses.decompiler.known_patterns.finder*), 508
OUTMAYBE (*angr.sim_type.PointerDisposition attribute*), 1466
output() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
output() (*angr.analyses.cfg.cfg_fast.CFGFast method*), 325
OVERFLOW_FP_RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
OVERFLOW_FP_RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64ABI0 attribute*), 976
OVERFLOW_FP_RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 967
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCARM attribute*), 977
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCARMHF attribute*), 977
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 971
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64ABI0 attribute*), 976
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973

OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 972
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCO32 attribute*), 980
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCPowerPC attribute*), 981
OVERFLOW_RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
OVERFLOW_RETURN_VAL (*angr.SimCC attribute*), 173
overlap() (*angr.knowledge_plugins.patches.PatchManager static method*), 1265
OverlappingFunctionsAnalysis (*class in angr.analyses.patchfinder*), 792
OverlayManager (*class in angr.analyses.decompiler.region_overlay*), 624
owner_of() (*angr.analyses.decompiler.region_overlay.OverlayManager method*), 624

## P

pack (*angr.sim_type.SimStruct property*), 1471
pack_arg_vvars() (*in module angr.utils.ail_serialization*), 1672
pack_graph() (*in module angr.utils.ail_serialization*), 1671
pack_ite_exprs() (*in module angr.utils.ail_serialization*), 1672
pack_static_buffers() (*in module angr.utils.ail_serialization*), 1672
pack_static_vvars() (*in module angr.utils.ail_serialization*), 1672
packed (*angr.sim_type.SimStruct property*), 1471
PACKED_ENTROPY_MIN_THRESHOLD (*angr.analyses.PackingDetector attribute*), 259
PACKED_ENTROPY_MIN_THRESHOLD (*angr.analyses.unpacker.packing_detector.PackingDetector attribute*), 922
PACKED_ENTROPY_MIN_THRESHOLD (*angr.analyses.unpacker.PackingDetector attribute*), 921
PACKED_MIN_BYTES (*angr.analyses.PackingDetector attribute*), 258
PACKED_MIN_BYTES (*angr.analyses.unpacker.packing_detector.PackingDetector attribute*), 922
PACKED_MIN_BYTES (*angr.analyses.unpacker.PackingDetector attribute*), 921
PackingDetector (*class in angr.analyses.unpacker.packing_detector*), 922
paffinity_mask (*angr.procedures.win32.GetProcessAffinityMask.GetProc attribute*), 1383
PageBase (*class in angr.storage.memory_mixins.paged_memory.pages.base*), 1647
PagedMemoryMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mix*), 1641
PagedMemoryMultiValueMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mul*), 1643
PAny (*class in angr.analyses.decompiler.known_patterns.dsl*), 492
ParameterTag (*class in angr.knowledge_plugins.key_definitions.tag*), 1261
parent (*angr.analyses.pathfinder.SimStateMarker attribute*), 794
parentop (*angr.analyses.disassembly.OperandPiece attribute*), 742
parents (*angr.state_plugins.history.SimStateHistory property*), 1562
parents() (*angr.storage.memory_mixins.paged_memory.pages.history_tra method*), 1648
parse() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed class method*), 760
parse() (*angr.serializable.Serializable class method*), 1440
parse_address() (*in module angr.analyses.decompiler.edits.resolve*), 446
parse_arg_vvars() (*in module angr.utils.ail_serialization*), 1672
parse_block() (*angr.analyses.disassembly.Disassembly method*), 743
parse_codegen() (*in module angr.analyses.decompiler.structured_codegen.c_serialize*), 690
parse_cpp_file() (*in module angr.sim_type*), 1477
parse_defns() (*in module angr.sim_type*), 1476
parse_eh4_scopetable() (*in module angr.analyses.cfg.pe_msvc_eh_structs*), 344
parse_file() (*in module angr.sim_type*), 1476
parse_format_string() (*in module angr.analyses.reaching_definitions.function_handler_library.stdio*), 846
parse_from_cmessage() (*angr.analyses.decompiler.clinic.Clinic class method*), 408
parse_from_cmessage() (*angr.analyses.decompiler.decompilation_cache.DecompilationCa class method*), 413
parse_from_cmessage() (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo class method*), 1176
parse_from_cmessage()

(*angr.knowledge_plugins.functions.function.Function class method*), 1213
parse_from_cmessage() (*angr.serializable.Serializable class method*), 1440
parse_from_cmsg() (*angr.knowledge_plugins.functions.function_parser.FunctionParser static method*), 1227
parse_funcinfo() (*in module angr.analyses.cfg.pe_msvc_eh_structs*), 343
parse_gdb_xml_file() (*in module angr.procedures.definitions.parse_syscalls_from_local_system*), 1334
parse_graph() (*in module angr.utils.ail_serialization*), 1671
parse_handler_array() (*in module angr.analyses.cfg.pe_msvc_eh_structs*), 345
parse_header() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed class method*), 760
parse_ite_exprs() (*in module angr.utils.ail_serialization*), 1672
parse_known_patterns() (*in module angr.analyses.decompiler.decompilation_options*), 414
parse_module() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed method*), 759
parse_modules() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed method*), 759
PARSE_MORE_MODULES (*angr.analyses.flirt.consts.FlirtParseFlag attribute*), 755
PARSE_MORE_MODULES_WITH_SAME_CRC (*angr.analyses.flirt.consts.FlirtParseFlag attribute*), 755
PARSE_MORE_PUBLIC_NAMES (*angr.analyses.flirt.consts.FlirtParseFlag attribute*), 755
parse_node() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed static method*), 760
parse_public_function() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed method*), 759
PARSE_READ_REFERENCED_FUNCTIONS (*angr.analyses.flirt.consts.FlirtParseFlag attribute*), 755
PARSE_READ_TAIL_BYTES (*angr.analyses.flirt.consts.FlirtParseFlag attribute*), 755
parse_referenced_functions() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed method*), 759
parse_signature() (*in module angr.sim_type*), 1476
parse_stack_pointer() (*in module angr.analyses.variable_recovery.variable_recovery_fast*), 926
parse_static_buffers() (*in module angr.utils.ail_serialization*), 1672
parse_static_vvars() (*in module angr.utils.ail_serialization*), 1672
parse_subtree() (*in module angr.analyses.decompiler.structured_codegen.c_serialize*), 690
parse_tail_bytes() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed method*), 759
parse_tree() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed method*), 759
parse_try_block_map() (*in module angr.analyses.cfg.pe_msvc_eh_structs*), 346
parse_type() (*in module angr.sim_type*), 1476
parse_type_with_name() (*in module angr.sim_type*), 1477
parse_types() (*in module angr.sim_type*), 1476
parse_unistd_include_header() (*in module angr.procedures.definitions.parse_syscalls_from_local_system*), 1334
parse_unwind_map() (*in module angr.analyses.cfg.pe_msvc_eh_structs*), 344
parse_variable_addr() (*angr.analyses.decompiler.clinic.Clinic method*), 407
parse_variant_mask() (*angr.analyses.flirt.flirt_sig.FlirtSignatureParsed static method*), 760
ParseContext (*class in angr.analyses.decompiler.structured_codegen.c_serialize*), 689
parsedcprotos2py() (*in module angr.utils.library*), 1691
ParsedInstruction (*class in angr.analyses.data_dep.sim_act_location*), 361
ParseDouble (*class in angr.procedures.java_lang.double*), 1348
PartialConstantExprRewriter (*class in angr.analyses.decompiler.ail_simplifier*), 397
partition_templates() (*in module angr.analyses.decompiler.known_patterns.registry*), 525
pass_to_name() (*in module angr.analyses.decompiler.optimization_pass_registry*), 540
PAssign (*class in angr.analyses.decompiler.known_patterns.dsl*), 502
Patch (*class in angr.knowledge_plugins.patches*), 1264
patch_addrs() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
patched_entry_state (*angr.knowledge_plugins.patches.PatchManager property*), 1266

PatchedOutFunctionality (*class in angr.analyses.patchfinder*), 793
PatchFinderAnalysis (*class in angr.analyses.patchfinder*), 793
PatchManager (*class in angr.knowledge_plugins.patches*), 1264
path() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 918
path_between() (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink method*), 349
path_to_graph() (*angr.rust.analyses.rust_calling_convention.pathfinder static method*), 1409
Pathfinder (*class in angr.analyses.pathfinder*), 794
Pathfinder (*class in angr.rust.analyses.rust_calling_convention.pathfinder*), 1409
PathTerminator (*class in angr.procedures.stubs.PathTerminator*), 1377
PathUnreachableError, 1081
pattern_anchor_key() (*in module angr.analyses.decompiler.known_patterns.dsl*), 506
PatternContext (*class in angr.analyses.decompiler.known_patterns.context*), 488
PatternExpr (*class in angr.analyses.decompiler.known_patterns.dsl*), 492
PatternGate (*class in angr.analyses.decompiler.known_patterns.gating*), 511
PatternGenerationError, 515
PatternGenerator (*class in angr.analyses.decompiler.known_patterns.generator*), 515
PatternMatchNode (*class in angr.rust.structuring.structurer_nodes*), 1436
PatternMatchSimplifier (*class in angr.rust.optimization_passes.pattern_match_simplifier*), 1426
PatternMatchWalker (*class in angr.rust.optimization_passes.pattern_match_simplifier*), 1426
PatternNode (*class in angr.analyses.decompiler.known_patterns.dsl*), 491
PatternParam (*class in angr.analyses.decompiler.known_patterns.pattern*), 519
patterns_for() (*in module angr.analyses.decompiler.known_patterns.registry*), 525
PatternStmt (*class in angr.analyses.decompiler.known_patterns.dsl*), 502
PBinOp (*class in angr.analyses.decompiler.known_patterns.dsl*), 494
PBlockPat (*class in angr.analyses.decompiler.known_patterns.dsl*), 505
pc_actions_ADC() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_ADCX() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_ADD() (*in module angr.engines.vex.claripy.ccall*), 1068
pc_actions_ADOX() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_BLSI() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_BLSMSK() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_BLSR() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_cond_CondB() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondBE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondL() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondLE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondNBE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondNLE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondNZ() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondS() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_cond_CondZ() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_DEC() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_INC() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_LOGIC() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_op_ADD() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_op_DEC() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_op_INC() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_op_LOGIC() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_op_SHL() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_op_SHR() (*in module angr.engines.vex.claripy.ccall*), 1070

pc_actions_op_SUB() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_ROL() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_ROR() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_SBB() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_SHL() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_SHR() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_SMUL() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_SMULQ() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_SUB() (*in module angr.engines.vex.claripy.ccall*), 1069
pc_actions_SUB_CondB() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_SUB_CondBE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_SUB_CondL() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_SUB_CondLE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_SUB_CondNBE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_SUB_CondNLE() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_SUB_CondNZ() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_SUB_CondZ() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_actions_UMUL() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_actions_UMULQ() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_calculate_condition() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_calculate_condition_simple() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_calculate_rdata_all() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_calculate_rdata_all_WRK() (*in module angr.engines.vex.claripy.ccall*), 1070
pc_calculate_rdata_c() (*in module angr.engines.vex.claripy.ccall*), 1071
pc_make_rdata() (*in module angr.engines.vex.claripy.ccall*), 1068
pc_make_rdata_if_necessary() (*in module angr.engines.vex.claripy.ccall*), 1068
pc_preamble() (*in module angr.engines.vex.claripy.ccall*), 1068
PCall (*class in angr.analyses.decompiler.known_patterns.dsl*), 497
PCallResult (*class in angr.analyses.decompiler.known_patterns.dsl*), 497
PCallStmt (*class in angr.analyses.decompiler.known_patterns.dsl*), 503
PChoice (*class in angr.analyses.decompiler.known_patterns.dsl*), 502
pcode (*angr.block.Block property*), 961
pcode_block_lifter() (*angr.project.Project method*), 1392
PcodeBasicBlockLifter (*class in angr.engines.pcode.lifter*), 1043
PCodeBlock (*class in angr.block*), 960
PcodeDisassemblerBlock (*class in angr.engines.pcode.lifter*), 1037
PcodeDisassemblerInsn (*class in angr.engines.pcode.lifter*), 1037
PcodeEmulatorMixin (*class in angr.engines.pcode.emulate*), 1036
PCodeInsn (*class in angr.block*), 960
PCodeIRSBConverter (*class in angr.ailment.converter_pcode*), 217
PcodeLifter (*class in angr.engines.pcode.lifter*), 1043
PcodeLifterEngineMixin (*class in angr.engines.pcode.lifter*), 1044
PCondJump (*class in angr.analyses.decompiler.known_patterns.dsl*), 504
PConst (*class in angr.analyses.decompiler.known_patterns.dsl*), 493
PConv (*class in angr.analyses.decompiler.known_patterns.dsl*), 495
PDefOf (*class in angr.analyses.decompiler.known_patterns.dsl*), 498
PEDelayImportDirectory (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
PEDelayImportDirectory (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
PEDelayImportDirectory (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
peek_fn (*angr.analyses.decompiler.known_patterns.dsl.MatchCtx attribute*), 491
peek_input() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
peek_output() (*angr.state_plugins.cgc.SimStateCGC method*), 1545
peephole_optimize_expr() (*in module angr.analyses.decompiler.utils*), 724
peephole_optimize_exprs() (*in module angr.analyses.decompiler.utils*), 724

peephole_optimize_multistmts() (*in module angr.analyses.decompiler.utils*), 725
peephole_optimize_stmts() (*in module angr.analyses.decompiler.utils*), 724
PeepholeOptimizationBundle (*class in angr.analyses.decompiler.block_simplifier*), 399
PeepholeOptimizationExprBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 604
PeepholeOptimizationMultiStmtBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 603
PeepholeOptimizationStmtBase (*class in angr.analyses.decompiler.peephole_optimizations.base*), 602
PEExportDirectory (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
PEExportDirectory (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
PEExportDirectory (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
PEImportDirectory (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1178
PEImportDirectory (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
PEImportDirectory (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
PendingJob (*class in angr.analyses.cfg.cfg_emulated*), 310
PendingJob (*class in angr.analyses.vfg*), 932
PendingJobs (*class in angr.analyses.cfg.cfg_fast*), 315
perform_call() (*angr.ail_callable.AILCallable method*), 208
perform_call() (*angr.analyses.identifier.custom_callable.CustomCallable method*), 770
perform_call() (*angr.callable.Callable method*), 962
perform_def() (*angr.analyses.decompiler.ssailification.traversal_engine.SimEngineSSATraversal method*), 657
perm_exec (*angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin property*), 1651
perm_read (*angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin property*), 1651
perm_write (*angr.storage.memory_mixins.paged_memory.pages.permissions_mixin.PermissionsMixin property*), 1651
permissions() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 1634
permissions() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1639
permissions() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1642
PermissionsMixin (*class in angr.storage.memory_mixins.paged_memory.pages.permissions_mixin*), 1651
Permissive (*angr.exploration_techniques.tracer.TracingMode attribute*), 1107
perror (*class in angr.procedures.libc.perror*), 1358
persistent_id() (*angr.vaults.VaultPickler method*), 1704
persistent_load() (*angr.vaults.VaultUnpickler method*), 1704
PExtract (*class in angr.analyses.decompiler.known_patterns.dsl*), 495
PField (*class in angr.analyses.decompiler.known_patterns.dsl*), 499
PGraphPat (*class in angr.analyses.decompiler.known_patterns.dsl*), 505
Phi (*class in angr.ailment.expression*), 221
PhiVar (*class in angr.protos.variables_pb2*), 1405
phi_assignment_get_src() (*in module angr.utils.ssa*), 1698
PhiAssignmentCollector (*class in angr.analyses.decompiler.dephication.seqnode_dephication*), 420
PhoenixStructurer (*class in angr.analyses.decompiler.structuring.phoenix*), 717
PicklableLock (*class in angr.misc.picklable_lock*), 1317
PicklableRLock (*class in angr.misc.picklable_lock*), 1317
PickledState (*class in angr.exploration_techniques.spiller_db*), 1105
PickledStatesBase (*class in angr.exploration_techniques.spiller*), 1104
PickledStatesDb (*class in angr.exploration_techniques.spiller*), 1105
PickledStatesList (*class in angr.exploration_techniques.spiller*), 1104
PITE (*class in angr.analyses.decompiler.known_patterns.dsl*), 501
PLAIN (*angr.knowledge_plugins.CommentKind attribute*), 1133
PLAIN (*angr.knowledge_plugins.comments.CommentKind attribute*), 1189
Platform (*in module angr.engines.vex.claripy.ccall*), 1068
platforms (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 457
platforms (*angr.analyses.decompiler.known_patterns.KnownPatternTemplate attribute*), 463
platforms (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521
platforms (*angr.analyses.decompiler.known_patterns.templates.KnownPattern attribute*), 532
platforms (*angr.analyses.decompiler.KnownPattern attribute*), 379

PLATFORMS (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePointerSaveSimplifier attribute*), 560
PLATFORMS (*angr.analyses.decompiler.optimization_passes.call_stmt_rewriter.CallStatementRewriter attribute*), 560
PLATFORMS (*angr.analyses.decompiler.optimization_passes.code_motion.CodeMotionOptimizationPass attribute*), 561
PLATFORMS (*angr.analyses.decompiler.optimization_passes.condition_constprop.ConditionConstantPropagation attribute*), 562
PLATFORMS (*angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDereferencesSimplifier attribute*), 563
PLATFORMS (*angr.analyses.decompiler.optimization_passes.const_prop_reverter.ConstantPropagationRevert attribute*), 563
PLATFORMS (*angr.analyses.decompiler.optimization_passes.engine_base.SimEngineStructuredDirtyingBase attribute*), 564
PLATFORMS (*angr.analyses.decompiler.optimization_passes.engine_base.SimEngineStructuredRemovingBase attribute*), 565
PLATFORMS (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.BasePtrSaveSimplifier attribute*), 566
PLATFORMS (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.OpTransformationSimplifier attribute*), 566
PLATFORMS (*angr.analyses.decompiler.optimization_passes.flag_setting_cond_simplifier.FlagSettingCondSimplifier attribute*), 567
PLATFORMS (*angr.analyses.decompiler.optimization_passes.inlined_memset_finder.InlinedMemsetFinder attribute*), 568
PLATFORMS (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier attribute*), 569
PLATFORMS (*angr.analyses.decompiler.optimization_passes.lazy_simplifier.LazySimplifier attribute*), 570
PLATFORMS (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 571
PLATFORMS (*angr.analyses.decompiler.optimization_passes.multi_simplifier.MultiSimplifier attribute*), 572
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 573
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pass.StructuringOptimizationPass attribute*), 574
PLATFORMS (*angr.analyses.decompiler.optimization_passes.peephole_optimizations.PeepholeOptimizationExpr attribute*), 575
PLATFORMS (*angr.analyses.decompiler.optimization_passes.peephole_optimizations.PeepholeOptimizationMultiExpr attribute*), 576
PLATFORMS (*angr.analyses.decompiler.optimization_passes.peephole_optimizations.PeepholeOptimizationStmt attribute*), 576
PLATFORMS (*angr.analyses.decompiler.optimization_passes.register_save_simplifier.RegisterSaveSimplifier attribute*), 577
PLATFORMS (*angr.analyses.decompiler.optimization_passes.return_duplicator_base.CodeDuplicationOptimization attribute*), 578
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 578
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanaryStoreSimplifier attribute*), 579
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanaryVariableSimplifier attribute*), 579
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanaryXorSimplifier attribute*), 580
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanaryZeroSimplifier attribute*), 580
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanaryZeroStoreSimplifier attribute*), 581
PLATFORMS (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.StrcpyStrcatRecognizer attribute*), 581
PLATFORMS (*angr.analyses.decompiler.optimization_passes.inlined_strlen_simplifier.StrlenSimplifier attribute*), 581
PLATFORMS (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simplifier.WcscpySimplifier attribute*), 581
PLATFORMS (*angr.analyses.decompiler.optimization_passes.InlinedMemcpyRewriter attribute*), 548
PLATFORMS (*angr.analyses.decompiler.optimization_passes.InlinedMemsetSimplifier attribute*), 548
PLATFORMS (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 549
PLATFORMS (*angr.analyses.decompiler.optimization_passes.InlinedStringTransformationSimplifier attribute*), 549
PLATFORMS (*angr.analyses.decompiler.optimization_passes.InlinedStrlenSimplifier attribute*), 549
PLATFORMS (*angr.analyses.decompiler.optimization_passes.InlinedWcscpySimplifier attribute*), 550
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 583
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ite_region_converter.ITERegionConverter attribute*), 583
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ITEExprConverter attribute*), 547
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 547
PLATFORMS (*angr.analyses.decompiler.optimization_passes.known_pattern_identifier.KnownPatternIdentifier attribute*), 583
PLATFORMS (*angr.analyses.decompiler.optimization_passes.KnownPatternCategory attribute*), 550
PLATFORMS (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier attribute*), 585
PLATFORMS (*angr.analyses.decompiler.optimization_passes.LoweredSwitchSimplifier attribute*), 551
PLATFORMS (*angr.analyses.decompiler.optimization_passes.mips_gp_setting_cond_simplifier.MipsGpSettingCondSimplifier attribute*), 585
PLATFORMS (*angr.analyses.decompiler.optimization_passes.MipsGpSettingSimplifier attribute*), 551
PLATFORMS (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModExprOperation attribute*), 586
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 552
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pass.OptimizationPass attribute*), 587
PLATFORMS (*angr.analyses.decompiler.optimization_passes.optimization_pass.StructuringOptimizationPass attribute*), 589
PLATFORMS (*angr.analyses.decompiler.optimization_passes.peephole_simplifier_multi_expr.PeepholeOptimizationMultiExpr attribute*), 590
PLATFORMS (*angr.analyses.decompiler.optimization_passes.register_save_simplifier.RegisterSaveSimplifier attribute*), 590
PLATFORMS (*angr.analyses.decompiler.optimization_passes.register_save_simplifier.RegisterSaveSimplifierAdvanced attribute*), 590
PLATFORMS (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifier attribute*), 553

PLATFORMS (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifierAdvanced attribute*), 553
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 591
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ret_dedupl[illegible].ReturnDeduplicator attribute*), 591
PLATFORMS (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSimplifier attribute*), 553
PLATFORMS (*angr.analyses.decompiler.optimization_passes.return_duplicator_high.ReturnDuplicatorHigh attribute*), 592
PLATFORMS (*angr.analyses.decompiler.optimization_passes.return_duplicator_low.ReturnDuplicatorLow attribute*), 593
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ReturnDeduplicator attribute*), 554
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHigh attribute*), 554
PLATFORMS (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorLow attribute*), 555
PLATFORMS (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 594
PLATFORMS (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifier attribute*), 556
PLATFORMS (*angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.StaticVVarRewriter attribute*), 596
PLATFORMS (*angr.analyses.decompiler.optimization_passes.StaticVVarRewriter attribute*), 556
PLATFORMS (*angr.analyses.decompiler.optimization_passes.switch_default_case_duplicator.SwitchDefaultCaseDuplicator attribute*), 596
PLATFORMS (*angr.analyses.decompiler.optimization_passes.switch_reused_entry_rewriter.SwitchReusedEntryRewriter attribute*), 597
PLATFORMS (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseDuplicator attribute*), 556
PLATFORMS (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryRewriter attribute*), 557
PLATFORMS (*angr.analyses.decompiler.optimization_passes.tag_slicer.TagSlicer attribute*), 597
PLATFORMS (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 558
PLATFORMS (*angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier.WinStackCanarySimplifier attribute*), 598
PLATFORMS (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier attribute*), 558
PLATFORMS (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 598
PLATFORMS (*angr.analyses.decompiler.optimization_passes.X86GccGetPcSimplifier attribute*), 558
PLATFORMS (*angr.analyses.deobfuscator.string_obf_opt_passes.StringObfType3Rewriter attribute*), 739
PLATFORMS (*angr.analyses.deobfuscator.StringObfType3Rewriter attribute*), 733
PLATFORMS (*angr.rust.optimization_passes.cleanup_code_remover.CleanupCodeRemover attribute*), 1421
PLATFORMS (*angr.rust.optimization_passes.combo_register_[illegible] attribute*), 1421
PLATFORMS (*angr.rust.optimization_passes.deref_coercion_simplifier.Deref[illegible] attribute*), 1422
PLATFORMS (*angr.rust.optimization_passes.deref_coercion_simplifier_uninl[illegible] attribute*), 1422
PLATFORMS (*angr.rust.optimization_passes.error_propagation_simplifier.Er[illegible] attribute*), 1423
PLATFORMS (*angr.rust.optimization_passes.function_prototype_inference.Fu[illegible] attribute*), 1423
PLATFORMS (*angr.rust.optimization_passes.[illegible]macro.format_macro_simplifier.[illegible] attribute*), 1424
PLATFORMS (*angr.rust.optimization_passes.outliners.string_cmp_outliner.St[illegible] attribute*), 1424
PLATFORMS (*angr.rust.optimization_passes.outliners.string_literal_outliner.[illegible] attribute*), 1425
PLATFORMS (*angr.rust.optimization_passes.outliners.string_outliner.StringO[illegible] attribute*), 1425
PLATFORMS (*angr.rust.optimization_passes.outliners.unwrap_outliner.Unwr[illegible] attribute*), 1425
PLATFORMS (*angr.rust.optimization_passes.outliners.vec_outliner.VecOutlin[illegible] attribute*), 1426
PLATFORMS (*angr.rust.optimization_passes.pattern_match_simplifier.Pattern[illegible] attribute*), 1426
PLATFORMS (*angr.rust.optimization_passes.pre_pattern_match_simplifier.P[illegible] attribute*), 1427
PLATFORMS (*angr.rust.optimization_passes.redundant_block_remover.Redu[illegible] attribute*), 1428
PLATFORMS (*angr.rust.optimization_passes.[illegible].RetExprRewr[illegible] attribute*), 1428
PLATFORMS (*angr.rust.optimization_passes.rust_calling_convention.RustCa[illegible] attribute*), 1428
PLATFORMS (*angr.rust.optimization_passes.security_check_remover.Security[illegible] attribute*), 1428
PLATFORMS (*angr.rust.optimization_passes.str_argument_simplifier.StrArgu[illegible] attribute*), 1429
PLATFORMS (*angr.rust.optimization_passes.struct_instantiation_simplifier.S[illegible] attribute*), 1430
PLATFORMS (*angr.rust.optimization_passes.struct_return_simplifier.StructR[illegible] attribute*), 1430
Pload (*class in angr.[illegible]ms.dsl*), 496
plugin_preset (*angr.misc.plugins.PluginHub property*), 1318
PluginHub (*class in angr.misc.plugins*), 1317
PluginPreset (*class in angr.misc.plugins*), 1319
plugins (*angr.sim_state.SimState property*), 1454
PluginVendor (*class in angr.misc.plugins*), 1320
Pointer (*class in angr.analyses.typehoon.typeconsts*), 909
Pointer32 (*class in angr.analyses.typehoon.typeconsts*), 910
Pointer64 (*class in angr.analyses.typehoon.typeconsts*), 910
PointerArithmeticRewriter (*class in angr.analyses.decompiler.structured_codegen.c*),

687
PointerArithmeticFixer (*class in angr.analyses.decompiler.structured_codegen.rust*), 710
PointerArray (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
PointerArray (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
PointerArray (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
PointerDisposition (*class in angr.sim_type*), 1466
PointerNaming (*class in angr.analyses.decompiler.semantic_naming.pointer_naming*), 651
PointerWrapper (*class in angr.calling_conventions*), 962
poll (*class in angr.procedures.posix.poll*), 1373
pop() (*angr.state_plugins.callstack.CallStack method*), 1543
pop() (*angr.state_plugins.globals.SimStateGlobals method*), 1551
pop_dirty() (*angr.analyses.stack_pointer_tracker.StackPointerTracker.State method*), 895
pop_from_backup() (*angr.state_plugins.trace_additions.CallReplayInfo method*), 1597
pop_job() (*angr.analyses.cfg.cfg_fast.PendingJobs method*), 316
pop_n() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 1104
pop_priv() (*angr.state_plugins.scratch.SimStateScratch method*), 1583
pop_stack_frame() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1636
populate() (*angr.sim_manager.SimulationManager method*), 1445
populate_model() (*in module angr.analyses.s_reaching_definitions.s_rda_model*), 888
pos (*angr.SimFileBase attribute*), 179
pos (*angr.storage.file.SimFileBase attribute*), 1610
PositionMapping (*class in angr.analyses.decompiler.structured_codegen.base*), 668
PositionMappingElement (*class in angr.analyses.decompiler.structured_codegen.base*), 668
PositionMappingEntry (*class in angr.protos.codegen_pb2*), 1401
PositionMappingMsg (*class in angr.protos.codegen_pb2*), 1401
PosixDevFS (*class in angr.state_plugins.posix*), 1579
PosixProcFS (*class in angr.state_plugins.posix*), 1579
PossibleObject (*class in angr.analyses.find_objects_static*), 746
POST_CALLSITES (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
post_dom (*angr.utils.graph.PostDominators property*), 1685
post_filter() (*angr.exploration_techniques.stub_stasher.StubStasher static method*), 1106
POST_FORGOTTEN (*angr.analyses.typehoon.simple_solver.FORGOTTEN attribute*), 901
POST_SSA_LEVEL1_SIMPLIFICATIONS (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
PostDominators (*class in angr.utils.graph*), 1685
PostStructuringPeepholeOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.peephole_simpli*, 589
pow() (*angr.engines.vex.claripy.irop.SimIROp static method*), 1074
pp() (*angr.ailment.Block method*), 210
pp() (*angr.ailment.block.Block method*), 215
pp() (*angr.analyses.datagraph_meta.DataGraphMeta method*), 361
pp() (*angr.analyses.ddg.DDG method*), 365
pp() (*angr.analyses.smc.TraceClassifier method*), 892
pp() (*angr.block.Block method*), 961
pp() (*angr.block.DisassemblerBlock method*), 959
pp() (*angr.engines.pcode.lifter.IRSB method*), 1040
pp() (*angr.knowledge_plugins.functions.function.Function method*), 1218
pp_constraints() (*angr.analyses.typehoon.typehoon.Typehoon method*), 914
pp_solution() (*angr.analyses.typehoon.typehoon.Typehoon method*), 914
pp_str() (*angr.analyses.typehoon.typeconsts.TypeConstant method*), 907
pp_str() (*angr.analyses.typehoon.typevars.Add method*), 916
pp_str() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 918
pp_str() (*angr.analyses.typehoon.typevars.Equivalence method*), 915
pp_str() (*angr.analyses.typehoon.typevars.Existence method*), 915
pp_str() (*angr.analyses.typehoon.typevars.Sub method*), 916
pp_str() (*angr.analyses.typehoon.typevars.Subtype method*), 915
pp_str() (*angr.analyses.typehoon.typevars.TypeConstraint method*), 914
pp_str() (*angr.analyses.typehoon.typevars.TypeVariable method*), 917
ppc32g_dirtyhelper_MFSPR_287() (*in module angr.engines.vex.heavy.dirty*), 1076
ppcg_dirtyhelper_MFTB() (*in module angr.engines.vex.heavy.dirty*), 1076

PPhi (*class in angr.analyses.decompiler.known_patterns.dsl*), 500
PRE_FORGOTTEN (*angr.analyses.typehoon.simple_solver.FORGOTTEN attribute*), 901
PRE_SSA_LEVEL0_FIXUPS (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
PRE_SSA_LEVEL1_SIMPLIFICATIONS (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
pre_test() (*angr.analyses.identifier.func.Func method*), 771
pread64 (*class in angr.procedures.posix.pread64*), 1373
preconstrain() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 1582
preconstrain_file() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 1583
preconstrain_flag_page() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 1583
pred (*angr.analyses.decompiler.known_patterns.dsl.PConst attribute*), 493
pred (*angr.analyses.decompiler.known_patterns.PConst attribute*), 469
predecessors (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
predecessors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 767
predecessors() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 835
predecessors() (*angr.codenode.CodeNode method*), 987
predecessors() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1185
predecessors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 1174
predicate() (*angr.analyses.s_reaching_definitions.s_rda_view.StackVarPredicate method*), 889
predicate() (*angr.analyses.s_reaching_definitions.s_rda_view.StackVarPredicate method*), 889
prep() (*angr.analyses.analysis.AnalysisFactory method*), 276
prep_tracer() (*angr.state_plugins.trace_additions.ChallRespInfo static method*), 1597
prep_tracer() (*angr.state_plugins.trace_additions.ZenPlugin static method*), 1598
prepare_call_state() (*angr.simos.simos.SimOS method*), 1489
prepare_function_symbol() (*angr.simos.linux.SimLinux method*), 1488
prepare_function_symbol() (*angr.simos.simos.SimOS method*), 1489
prepare_native_return_state() (*angr.engines.soot.engine.SootMixin static method*), 1047
prepare_native_return_state() (*in module angr.simos.javavm*), 1488
prepare_return_state() (*angr.engines.soot.engine.SootMixin static method*), 1047
PrePatternMatchSimplifier (*class in angr.rust.optimization_passes.pre_pattern_match_simplifier*), 1426
preprocess() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 903
pretty_this (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 677
prev_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
prev_chunk() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 1560
prev_size() (*angr.state_plugins.heap.heap_ptmalloc.PTChunk method*), 1560
print_all_chunks() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 1558
print_git_info() (*in module angr.misc.bug_report*), 1315
print_heap_state() (*angr.state_plugins.heap.heap_freelist.SimHeapFreelist method*), 1558
print_platform_info() (*in module angr.misc.bug_report*), 1315
print_python_info() (*in module angr.misc.bug_report*), 1315
print_timing_total() (*in module angr.utils.timing*), 1700
print_versions() (*in module angr.misc.bug_report*), 1315
PRINTABLES (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 321
PRINTABLES (*angr.analyses.cfg.CFGFast attribute*), 299
PRINTABLES (*angr.analyses.CFGFast attribute*), 242
printf (*class in angr.analyses.identifier.functions.printf*), 774
printf (*class in angr.procedures.libc.printf*), 1358
prioritize_functions() (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis method*), 354
PRIORITY (*angr.analyses.decompiler.semantic_naming.array_index_naming.ArrayIndexNaming attribute*), 647
PRIORITY (*angr.analyses.decompiler.semantic_naming.ArrayIndexNaming attribute*), 642
PRIORITY (*angr.analyses.decompiler.semantic_naming.boolean_naming.BooleanNaming attribute*), 647
PRIORITY (*angr.analyses.decompiler.semantic_naming.BooleanNaming attribute*), 643
PRIORITY (*angr.analyses.decompiler.semantic_naming.call_result_naming.

*attribute*), 648
PRIORITY (*angr.analyses.decompiler.semantic_naming.CallReturnNaming attribute*), 643
PRIORITY (*angr.analyses.decompiler.semantic_naming.naming_base.SemanticNamingBase attribute*), 648
PRIORITY (*angr.analyses.decompiler.semantic_naming.pointer_naming.PointerNaming attribute*), 651
PRIORITY (*angr.analyses.decompiler.semantic_naming.PointerNaming attribute*), 644
PRIORITY (*angr.analyses.decompiler.semantic_naming.region_naming.RegionNaming attribute*), 651
PRIORITY (*angr.analyses.decompiler.semantic_naming.RegionNaming attribute*), 644
PRIORITY (*angr.analyses.decompiler.semantic_naming.SemanticNamingBase attribute*), 645
PRIORITY (*angr.analyses.decompiler.semantic_naming.size_naming.SizeNaming attribute*), 652
PRIORITY (*angr.analyses.decompiler.semantic_naming.SizeNaming attribute*), 646
priv (*angr.state_plugins.scratch.SimStateScratch property*), 1583
Privileged (*angr.protos.primitives_pb2.Edge attribute*), 1403
PrivilegedPagingMixin (*class in angr.storage.memory_mixins.paged_memory.privileged_paging_mixin*), 1655
probably_identical (*angr.analyses.bindiff.FunctionDiff property*), 281
Procedure (*class in angr.analyses.reassembler*), 873
ProcedureChunk (*class in angr.analyses.reassembler*), 874
ProcedureEngine (*class in angr.engines.procedure*), 1046
ProcedureMixin (*class in angr.engines.procedure*), 1046
process() (*angr.analyses.decompiler.ssailification.rewriting_engine.SSARewriter method*), 654
process() (*angr.analyses.propagator.engine_base.SimEnginePropagatorBase method*), 796
process() (*angr.analyses.purity.engine.PurityEngineAIL method*), 803
process() (*angr.analyses.reaching_definitions.engine_ail.SimEngineRDAIL method*), 838
process() (*angr.analyses.reaching_definitions.engine_vex.SimEngineRDVEX method*), 838
process() (*angr.analyses.variable_recovery.irsb_scanner.VEXIRSBScanner method*), 925
process() (*angr.engines.ail.engine_light.SimEngineAILSimOS method*), 1003
process() (*angr.engines.light.engine.SimEngineLight method*), 1011
process() (*angr.engines.light.engine.SimEngineLightAIL method*), 1013
process() (*angr.engines.light.engine.SimEngineLightVEX method*), 1013
process() (*angr.engines.soot.expressions.base.SimSootExpr method*), 1050
process() (*angr.engines.soot.statements.base.SimSootStmt method*), 1054
process() (*angr.engines.successors.SuccessorsEngine method*), 1064
process() (*angr.engines.vex.light.slicing.VEXSlicingMixin method*), 1081
process_execfile() (*angr.region_loop_configuration_building*), 1115
process_procedure() (*angr.engines.procedure.ProcedureMixin method*), 1046
process_successors() (*angr.engines.successors.SuccessorsEngine method*), 1064
procs (*angr.procedures.win32.dynamic_loading.GetProcAddress property*), 1384
ProgramVariable (*class in angr.analyses.ddg*), 361
project (*angr.analyses.decompiler.known_patterns.GateContext attribute*), 456
project (*angr.analyses.decompiler.known_patterns.gating.GateContext attribute*), 511
project (*angr.analyses.decompiler.optimization_passes.optimization_pass property*), 587
project (*angr.analyses.disassembly.Value property*), 742
project (*angr.knowledge_plugins.cfg.cfg_model.CFGModel property*), 1166
project (*angr.knowledge_plugins.functions.function.Function property*), 1211
Project (*class in angr.project*), 1388
project_language() (*in module angr.calling_conventions*), 984
ProjectNSFEmbedPDFAR (*class in angr.mcp.rewriting*), 1304
ProjectSession (*class in angr.mcp.session*), 1307
prop_key (*angr.analyses.propagator.propagator.PropagatorAnalysis property*), 797
PROPAGATED (*angr.knowledge_plugins.callsite_prototypes.CallsitePrototype attribute*), 1150
PropagationManager (*class in angr.knowledge_plugins.propagations.propagation_manager*), 1270
PropagationModel (*class in angr.knowledge_plugins.propagations.propagation_model*), 1271
propagator_load_callback() (*angr.analyses.cfg.indirect_jump_resolvers.propagator_utils.PropagatorLoadCallback method*), 339
PropagatorAnalysis (*class in angr.analyses.propagator.propagator*), 797
PropagatorLoadCallback (*class in angr.analyses.cfg.indirect_jump_resolvers.propagator_utils*),

339
PropagatorState (*class in angr.knowledge_plugins.propagations.states*), 1272
PropagatorVEXState (*class in angr.knowledge_plugins.propagations.states*), 1275
PropValue (*class in angr.knowledge_plugins.propagations.propagation*), 1268
protect_stdio_from_forked_children() (*in module angr.utils.mp*), 1694
prototype (*angr.analyses.decompiler.structured_codegen.c.CFunction property*), 677
prototype (*angr.analyses.decompiler.structured_codegen.misc.CExternalFunction property*), 698
prototype (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
prototype (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
prototype (*angr.knowledge_plugins.functions.function.Function property*), 1211
prototype (*angr.rust.analyses.rust_calling_convention.rust_calling_convention.RustArgumentAnalysis property*), 1409
prototype() (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern method*), 522
prototype() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
prototype_for_call() (*in module angr.analyses.decompiler.known_patterns.apply*), 485
prototype_libname (*angr.knowledge_plugins.functions.function.Function property*), 1211
prototype_returnty (*angr.analyses.decompiler.structured_codegen.c.CFunction property*), 677
prototype_source (*angr.knowledge_plugins.functions.function.Function property*), 1211
PrototypeSource (*class in angr.knowledge_plugins.functions.function*), 1209
ProximityGraphAnalysis (*class in angr.analyses.proximity_graph*), 800
ProxiNodeTypes (*class in angr.analyses.proximity_graph*), 798
prune() (*angr.sim_manager.SimulationManager method*), 1445
pseudocode (*angr.knowledge_plugins.functions.function.Function property*), 1214
PStackField (*class in angr.analyses.decompiler.known_patterns.dsl*), 499
PStmtSeq (*class in angr.analyses.decompiler.known_patterns.dsl*), 504
PStore (*class in angr.analyses.decompiler.known_patterns.dsl*), 503
PTChunk (*class in angr.state_plugins.heap.heap_ptmalloc*), 1559
PTChunkIterator (*class in angr.state_plugins.heap.heap_ptmalloc*), 1560
pthread_cond_signal (*class in angr.procedures.posix.pthread*), 1373
pthread_create (*class in angr.procedures.posix.pthread*), 1373
pthread_mutex_lock (*class in angr.procedures.posix.pthread*), 1373
pthread_mutex_unlock (*class in angr.procedures.posix.pthread*), 1373
pthreadExitFunction (*class in angr.procedures.posix.pthread*), 1373
ptr (*angr.analyses.decompiler.known_patterns.CppRef attribute*), 455
ptr (*angr.analyses.decompiler.known_patterns.pattern.CppRef attribute*), 519
ptr_bits (*angr.analyses.decompiler.known_patterns.dsl.MatchCtx attribute*), 491
ptr_load_count (*angr.analyses.purity.engine.DataUsage attribute*), 801
ptr_store_count (*angr.analyses.purity.engine.DataUsage attribute*), 801
PUnaryOp (*class in angr.analyses.decompiler.known_patterns.dsl*), 494
pure_calls (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 458
pure_calls (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521
pure_calls (*angr.analyses.decompiler.KnownPattern attribute*), 370
PurityEngineAIL (*class in angr.analyses.purity.engine*), 803
push() (*angr.state_plugins.callstack.CallStack method*), 1543
push_priv() (*angr.state_plugins.scratch.SimStateScratch method*), 1583
push_stack_frame() (*angr.storage.memory_mixins.javavm_memory_mixin method*), 1636
put() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 895
putc (*in module angr.procedures.libc.fputc*), 1355
putc_unlocked (*in module angr.procedures.libc.fputc*), 1355
putchar (*class in angr.procedures.libc.putchar*), 1359
PutHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 336
puts (*class in angr.procedures.libc.puts*), 1359
PVar (*class in angr.analyses.decompiler.known_patterns.dsl*), 492
pwrite64 (*class in angr.procedures.posix.pwrite64*), 1374

pykind (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseHeadStatement attribute*), 714

## Q

qop_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1012
qualifies_for_implicit_cast() (*in module angr.analyses.decompiler.structured_codegen.c*), 670
qualifies_for_implicit_cast() (*in module angr.analyses.decompiler.structured_codegen.rust*), 691
qualifies_for_simple_cast() (*in module angr.analyses.decompiler.structured_codegen.c*), 670
qualifies_for_simple_cast() (*in module angr.analyses.decompiler.structured_codegen.rust*), 691
qualifies_for_width_cast() (*in module angr.analyses.decompiler.structured_codegen.c*), 670
qualifies_for_width_cast() (*in module angr.analyses.decompiler.structured_codegen.rust*), 691
quasi_topological_sort_nodes() (*angr.utils.graph.GraphUtils static method*), 1687
query() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1223
QueryPerformanceCounter (*class in angr.procedures.win32.sim_time*), 1386

## R

ram_usage (*angr.analyses.analysis.Analysis property*), 277
rand (*class in angr.procedures.libc.rand*), 1359
rand_bytes() (*in module angr.analyses.identifier.func*), 771
rand_str() (*in module angr.analyses.identifier.func*), 771
RANDOM (*angr.simos.windows.SecurityCookieInit attribute*), 1492
random (*class in angr.procedures.cgc.random*), 1323
Random (*class in angr.procedures.java_lang.math*), 1349
random (*class in angr.procedures.tracer.random*), 1381
randomize_procedures() (*angr.analyses.reassembler.Reassembler method*), 877
raw_graph (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 627
raw_graph_with_successors (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 627
raw_to_bv() (*angr.sim_state_plugins.sim_action_object.SimActionObject method*), 1587
rda_observe_callback() (*angr.knowledge_plugins.key_definitions.key_definition_manager. method*), 1249
RDAObserverControl (*class in angr.knowledge_plugins.key_definitions.key_definition_manager*) 1248
RDAStateInitializer (*class in angr.analyses.reaching_definitions.rd_initializer*), 857
reachable() (*angr.state_plugins.history.SimStateHistory method*), 1562
reached_fixedpoint() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 768
ReachingDefinitionsAnalysis (*class in angr.analyses.reaching_definitions.reaching_definitions*), 867
ReachingDefinitionsModel (*class in angr.knowledge_plugins.key_definitions.rd_model*), 1259
ReachingDefinitionsState (*class in angr.analyses.reaching_definitions.rd_state*), 859
READ (*angr.knowledge_plugins.variables.variable_access.VariableAccessSo attribute*), 1282
Read (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 1293
Read (*angr.knowledge_plugins.xrefs.XRefType attribute*), 1291
READ (*angr.state_plugins.sim_action.SimActionData attribute*), 1585
READ (*angr.state_plugins.SimActionData attribute*), 1504
Read (*class in angr.procedures.java_io.read*), 1337
read (*class in angr.procedures.posix.read*), 1374
read() (*angr.storage.file.SimFileBase method*), 1611
read() (*angr.storage.file.SimFileDescriptorBase method*), 1614
read() (*angr.storage.file.SimPackets method*), 1613
read_data() (*angr.storage.file.SimFileDescriptorBase method*), 1614
read_from() (*angr.knowledge_plugins.variables.variable_manager.Variab method*), 1284
read_max_2_bytes() (*in module angr.analyses.flirt.flirt_utils*), 761
read_msr() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
read_multiple_bytes() (*in module angr.analyses.flirt.flirt_utils*), 761
read_pos (*angr.storage.file.SimFileDescriptor property*), 1616
read_pos (*angr.storage.file.SimFileDescriptorBase property*), 1615

read_pos (*angr.storage.file.SimFileDescriptorDuplex property*), 1616
read_short() (*in module angr.analyses.flirt.flirt_utils*), 761
read_storage (*angr.storage.file.SimFileDescriptor property*), 1615
read_storage (*angr.storage.file.SimFileDescriptorBase property*), 1615
read_storage (*angr.storage.file.SimFileDescriptorDuplex property*), 1616
read_word() (*in module angr.analyses.flirt.flirt_utils*), 761
readdir (*class in angr.procedures.posix.readdir*), 1374
ReadFile (*class in angr.procedures.win32.file_handles*), 1384
readv (*class in angr.procedures.linux_kernel.iovec*), 1366
real_length() (*angr.utils.dynamic_dictlist.DynamicDictList method*), 1678
realloc (*class in angr.analyses.identifier.functions.skip_realloc*), 778
realloc (*class in angr.procedures.libc.realloc*), 1359
realloc() (*angr.state_plugins.heap.heap_libc.SimHeapLibc method*), 1559
reanalyze (*angr.analyses.decompiler.edits.Refresh attribute*), 427
reanalyze (*angr.analyses.decompiler.edits.results.Refresh attribute*), 449
reapply_options() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 670
reapply_options() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 685
reapply_options() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeGenerator method*), 707
Reassembler (*class in angr.analyses.reassembler*), 875
ReassemblerFailureNotice, 870
rebuild_callgraph() (*angr.knowledge_plugins.functions.function_manager.FunctionManager method*), 1224
rebuild_default_handler_funcs() (*angr.ailment.block_walker.AILBlockWalker class method*), 215
recall() (*angr.analyses.typehoon.simple_solver.ConstraintGraph method*), 902
receive (*class in angr.procedures.cgc.receive*), 1324
receive (*class in angr.procedures.tracer.receive*), 1381
receive_n2 (*class in angr.analyses.identifier.functions.skip_recv_n*), 779
receive_n3 (*class in angr.analyses.identifier.functions.skip_recv_n*), 779
receive_n4 (*class in angr.analyses.identifier.functions.skip_recv_n*), 778
receive_until (*class in angr.analyses.identifier.functions.recv_until*), 777
receive_until_fd (*class in angr.analyses.identifier.functions.recv_until*), 777
recent_actions (*angr.state_plugins.history.SimStateHistory property*), 1562
recent_constraints (*angr.state_plugins.history.SimStateHistory property*), 1562
recomputed_defs (*angr.analyses.decompiler.known_patterns.finder.KnownPatternFinder attribute*), 507
recomputed_defs (*angr.analyses.decompiler.known_patterns.KnownPatternMatch attribute*), 461
recomputed_defs (*angr.analyses.decompiler.KnownPatternMatch attribute*), 383
reconstrain() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 1583
record_memory_data_addr() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
record_state() (*angr.errors.SimError method*), 1083
record_variable() (*angr.knowledge_plugins.variables.variable_manager method*), 1284
recover_calling_conventions() (*in module angr.mcp.server*), 1303
recover_edge_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 409
recover_edge_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 409
recover_reaching_conditions() (*angr.analyses.decompiler.condition_processor.ConditionProcessor method*), 409
RECOVER_VARIABLES (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
recurse_analysis() (*angr.analyses.reaching_definitions.function_handler.FunctionHandler method*), 844
RecursiveMerger (*class in angr.analyses.typehoon.simple_solver*), 900
RecursiveStructurer (*class in angr.analyses.decompiler.structuring.recursive_structurer*), 718
recv (*class in angr.procedures.posix.recv*), 1374
recvfrom (*class in angr.procedures.posix.recvfrom*), 1374
red (*angr.misc.ansi.BackgroundColor attribute*), 1314
red (*angr.misc.ansi.Color attribute*), 1314
redecompile (*angr.analyses.decompiler.edits.Refresh attribute*), 427
redecompile (*angr.analyses.decompiler.edits.results.Refresh attribute*), 449
redefine_locals (*angr.analyses.reaching_definitions.function_handler.FunctionHandler attribute*), 840
redefine_locals (*angr.analyses.reaching_definitions.FunctionCallData*

*attribute*), 807
Redirect (*class in angr.procedures.stubs.Redirect*), 1377
RedundantBlockRemover (*class in angr.rust.optimization_passes.redundant_block_remover*), 1428
RedundantLabelRemover (*class in angr.analyses.decompiler.redundant_label_remover*), 623
RedundantStackVariable (*class in angr.analyses.binary_optimizer*), 279
RefcountMixin (*class in angr.storage.memory_mixins.paged_memory.pages.refcount_mixin*), 1652
REFERENCE (*angr.knowledge_plugins.variables.variable_access.VariableAccessSort attribute*), 1282
reference_at() (*angr.knowledge_plugins.variables.variable_manager.VariableManager method*), 1284
reference_to (*angr.analyses.purity.engine.DataSource attribute*), 801
reference_values() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
reference_variable() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
reference_variable_offset() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
refine() (*angr.calling_conventions.SimFunctionArgument method*), 964
refine() (*angr.calling_conventions.SimLyingRegArg method*), 971
refine() (*angr.calling_conventions.SimRegArg method*), 964
refine() (*angr.calling_conventions.SimStackArg method*), 965
refine_locs_with_struct_type() (*in module angr.calling_conventions*), 963
reflow_types() (*in module angr.analyses.decompiler.edits.ops*), 443
reflow_variable_types() (*angr.analyses.decompiler.decompiler.Decompiler method*), 415
Refresh (*class in angr.analyses.decompiler.edits.results*), 449
refresh() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 673
reg (*angr.analyses.stack_pointer_tracker.OffsetVal property*), 894
REG (*angr.state_plugins.sim_action.SimAction attribute*), 1584
REG (*angr.state_plugins.SimAction attribute*), 1503
reg() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 1242
reg_concrete() (*angr.sim_state.SimState method*), 1455
reg_deps (*angr.state_plugins.sim_action.SimAction property*), 1584
reg_deps (*angr.state_plugins.sim_action.SimActionData property*), 1585
reg_load() (*angr.analyses.decompiler.optimization_passes.inlined_string_ method*), 579
reg_name (*angr.knowledge_plugins.key_definitions.definition.DefinitionMa attribute*), 1245
reg_offset (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariabl property*), 1244
REG_READ (*angr.state_plugins.inspect.EventType attribute*), 1565
reg_read_callback() (*angr.analyses.cfg.indirect_jump_resolvers.constant_value_mana method*), 738
reg_read_condition (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_read_condition (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_read_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_read_endness (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_read_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_read_expr (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_read_length (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_read_length (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_read_offset (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_read_offset (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_size (*angr.analyses.data_dep.dep_nodes.RegDepNode property*), 360
reg_store() (*angr.analyses.decompiler.optimization_passes.inlined_strin method*), 579
REG_WRITE (*angr.state_plugins.inspect.EventType attribute*), 1565
reg_write_condition (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_write_condition (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_write_endness (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_write_endness (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_write_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566

reg_write_expr (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_write_length (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_write_length (*angr.state_plugins.InspectAttrs attribute*), 1498
reg_write_offset (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
reg_write_offset (*angr.state_plugins.InspectAttrs attribute*), 1498
RegDepNode (*class in angr.analyses.data_dep.dep_nodes*), 360
regenerate_text() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 670
regenerate_text() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 685
regenerate_text() (*angr.analyses.decompiler.structured_codegen.dwarf_import.DwarfImportSourceCode method*), 691
regenerate_text() (*angr.analyses.decompiler.structured_codegen.must_structure.MustStructuredCodeGenerator method*), 707
region (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.RegionMetaMixin property*), 1662
region_ids (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.RegionMetaMixin property*), 1661
RegionBound (*class in angr.analyses.decompiler.region_overlay*), 623
RegionCategoryMixin (*class in angr.storage.memory_mixins.regioned_memory.region_category_mixin*), 1659
RegionDescriptor (*class in angr.storage.memory_mixins.regioned_memory.region_meta_mixin*), 1660
RegionedAddressConcretizationMixin (*class in angr.storage.memory_mixins.regioned_memory.regioned_address_concretization_mixin*), 1662
RegionedMemory (*class in angr.storage.memory_mixins*), 1627
RegionedMemoryMixin (*class in angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin*), 1663
RegionIdentifier (*class in angr.analyses.decompiler.region_identifier*), 623
RegionLoopCounterNaming (*class in angr.analyses.decompiler.semantic_naming.region_loop_counter_naming*), 651
RegionMap (*class in angr.storage.memory_mixins.regioned_memory.region_meta_mixin*), 1660
RegionNamingBase (*class in angr.analyses.decompiler.semantic_naming.naming_base*), 649
RegionObject (*class in angr.keyed_region*), 1115
RegionOverlay (*class in angr.analyses.decompiler.region_overlay*), 625
RegionOverlayGraph (*class in angr.analyses.decompiler.region_overlay*), 631
regions (*angr.analyses.cfg.cfb.CFBlanket property*), 305
regions (*angr.analyses.cfg.cfg_base.CFGBase property*), 309
RegionSimplifier (*class in angr.analyses.decompiler.region_simplifiers.region_simplifier*), 638
RegionWalker (*class in angr.analyses.decompiler.region_walker*), 642
Register (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 359
Register (*angr.analyses.data_dep.DepNodeTypes attribute*), 356
register (*angr.analyses.disassembly.RegisterOperand property*), 742
REGISTER (*angr.analyses.reaching_definitions.AtomKind attribute*), 807
REGISTER (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 1241
REGISTER (*angr.knowledge_plugins.variables.variable_manager.VariableType attribute*), 1283
REGISTER (*angr.knowledge_plugins.variables.VariableType attribute*), 1281
Register (*class in angr.ailment.expression*), 221
Register (*class in angr.analyses.disassembly*), 742
Register (*class in angr.analyses.stack_pointer_tracker*), 894
Register (*class in angr.knowledge_plugins.key_definitions.atoms*), 1244
register() (*angr.knowledge_plugins.key_definitions.atoms.Atom static method*), 1243
register() (*angr.procedures.win32.dynamic_loading.LoadLibraryA method*), 1383
register() (*cle.backends.externs.simdata.Initializer method*), 1694
register_all() (*in module angr.analyses.decompiler.structured_codegen.c_serialize*), 690
register_analysis() (*in module angr.analyses.analysis*), 277
register_bool_option() (*angr.sim_state_options.SimStateOptions method*), 1459
register_callbacks() (*angr.analyses.variable_recovery.variable_recovery.VariableRecovery method*), 925
register_data_reference() (*angr.analyses.reassembler.Reassembler method*), 876

register_default() (*angr.knowledge_plugins.plugin.KnowledgeBasePlugin static method*), 1266
register_default() (*angr.misc.plugins.PluginHub class method*), 1317
register_default() (*angr.state_plugins.plugin.SimStatePlugin class method*), 1579
register_default_cc() (*in module angr.calling_conventions*), 983
register_default_cc_by_language() (*in module angr.calling_conventions*), 983
register_function_analysis() (*angr.analyses.vfg.CallAnalysis method*), 933
register_get() (*angr.analyses.decompiler.ssailification.traversal_engine.SimEngineSSATraversal method*), 657
register_instruction_reference() (*angr.analyses.reassembler.Reassembler method*), 876
register_kernel_types() (*in module angr.utils.library*), 1691
register_optimization_pass() (*in module angr.analyses.decompiler.optimization_passes*), 558
register_option() (*angr.sim_state_options.SimStateOptions class method*), 1459
register_pattern_template() (*in module angr.analyses.decompiler.known_patterns.registry*), 524
register_pcode_arch_default_cc() (*in module angr.engines.pcode.cc*), 1036
register_plugin() (*angr.knowledge_base.KnowledgeBase method*), 1119
register_plugin() (*angr.misc.plugins.PluginHub method*), 1318
register_plugin() (*angr.misc.plugins.PluginVendor method*), 1320
register_preset() (*angr.misc.plugins.PluginHub class method*), 1318
register_read() (*angr.analyses.calling_convention.fact_collector.FactCollectorState method*), 287
register_read() (*angr.analyses.fcp.fcp.FCPState method*), 745
register_read_undo() (*angr.analyses.calling_convention.fact_collector.FactCollectorState method*), 287
register_results() (*angr.knowledge_plugins.propagations.states.PropagationState method*), 1276
register_set() (*angr.analyses.decompiler.ssailification.traversal_engine.SimEngineSSATraversal method*), 658
register_syscall_cc() (*in module angr.calling_conventions*), 984
register_types() (*in module angr.sim_type*), 1476
register_unify() (*angr.analyses.decompiler.ssailification.traversal_state.TraversalState method*), 659
register_uses() (*angr.analyses.reaching_definitions.rd_state.ReachingDefi property*), 862
register_variable() (*angr.state_plugins.solver.SimSolver method*), 1588
register_written() (*angr.analyses.calling_convention.fact_collector.Fa method*), 287
register_written() (*angr.analyses.fcp.fcp.FCPState method*), 745
RegisterAnnotation (*class in angr.knowledge_plugins.propagations.states*), 1274
RegisterComparisonAnnotation (*class in angr.knowledge_plugins.propagations.states*), 1275
RegisterExpressionCounter (*class in angr.analyses.decompiler.counters.expression_counters*), 412
RegisterInitializerHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 336
RegisterOffset (*class in angr.engines.light.data*), 1010
RegisterOperand (*class in angr.analyses.disassembly*), 742
RegisterReallocation (*class in angr.analyses.binary_optimizer*), 279
registers (*angr.analyses.reaching_definitions.rd_state.ReachingDefinition property*), 862
RegisterSaveAreaSimplifier (*class in angr.analyses.decompiler.optimization_passes.register_save_area* 590
RegisterSaveAreaSimplifierAdvanced (*class in angr.analyses.decompiler.optimization_passes.register_save_area* 590
RegisterValue (*class in angr.state_plugins.unicorn_engine*), 1599
RegisterVariable (*class in angr.protos.variables_pb2*), 1405
RegOffsetAnnotation (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 335
RegVVarPredicate (*class in angr.analyses.s_reaching_definitions.s_rda_view*), 888
rehook_symbol() (*angr.project.Project method*), 1391
Reinterpret (*class in VEXStatement.expression*), 221
ReinterpretAs (*class in angr.analyses.typeSSATraversal.errors*), 920
related_function_addr (*angr.storage.memory_mixins.regioned_memory.region_meta_mi property*), 1662
relativize() (*angr.storage.memory_mixins.regioned_memory.region_da method*), 1661
release() (*angr.misc.picklable_lock.PicklableLock*

*method*), 1317
release() (*angr.rustylib.SegmentList method*), 1439
release() (*angr.state_plugins.heap.heap_brk.SimHeapBrk method*), 1556
release_plugin() (*angr.knowledge_base.KnowledgeBase method*), 1119
release_plugin() (*angr.misc.plugins.PluginHub method*), 1319
release_plugin() (*angr.misc.plugins.PluginVendor method*), 1320
release_shared() (*angr.storage.memory_mixins.paged_memory.pages.refcount_mixin.RefcountMixin method*), 1652
ReleaseArrayElements (*class in angr.procedures.java_jni.array_operations*), 1339
ReleaseStringUTFChars (*class in angr.procedures.java_jni.string_operations*), 1347
relink_typerefs() (*in module angr.utils.types*), 1702
reload_format() (*angr.analyses.disassembly.Instruction method*), 741
reload_solver() (*angr.state_plugins.solver.SimSolver method*), 1588
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 670
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 685
reload_variable_types() (*angr.analyses.decompiler.structured_codegen.rust.RustStructuredCodeGenerator method*), 708
relocatable (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.RegOffsetAnnotation attribute*), 335
relocatable (*angr.analyses.data_dep.data_dependency_analysis.VexVariableAnnotation attribute*), 358
relocatable (*angr.analyses.decompiler.condition_processor.ConditionExpression attribute*), 408
relocatable (*angr.analyses.variable_recovery.annotations.StackLocationAnnotation attribute*), 923
relocatable (*angr.analyses.variable_recovery.annotations.VariableSourceAnnotation attribute*), 923
relocatable (*angr.analyses.variable_recovery.variable_recovery_base.VariableAnnotation attribute*), 927
relocatable (*angr.knowledge_plugins.key_definitions.live_definitions.DefinitionAnnotation attribute*), 1249
relocatable (*angr.knowledge_plugins.propagations.states.RegisterAnnotation attribute*), 1274
relocatable (*angr.knowledge_plugins.propagations.states.RegisterOffsetAnnotation attribute*), 1275
relocatable (*angr.storage.memory_mixins.address_concretization_mixin.MultiwriteAnnotation attribute*), 1633
Relocation (*class in angr.analyses.reassembler*), 874
relocations (*angr.analyses.reassembler.Reassembler property*), 875
remote_chase_fn (*angr.analyses.decompiler.known_patterns.dsl.MatchCt attribute*), 491
remote_defs (*angr.analyses.decompiler.known_patterns.dsl.MatchState attribute*), 490
remove() (*angr.sim_state_options.SimStateOptions method*), 1458
remove_block() (*angr.rust.mixins.cfg_transformation_mixin.CFGTransfo method*), 1418
remove_bookmark() (*angr.knowledge_plugins.bookmarks.Bookmarks method*), 1149
remove_breakpoint() (*angr.emulator.Emulator method*), 995
remove_breakpoint() (*angr.state_plugins.inspect.SimInspector method*), 1572
remove_cgc_attachments() (*angr.analyses.reassembler.Reassembler method*), 877
remove_claripy_bool_asts() (*angr.analyses.decompiler.condition_processor.ConditionProcess method*), 409
remove_cycles() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 313
remove_edge() (*angr.analyses.cfg.cfg_base.CFGBase method*), 309
remove_edge() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1184
remove_edge_by_key() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1184
remove_edge_with_successors_only() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 629
remove_edges_in_ail_graph() (*in module angr.analyses.decompiler.utils*), 727
remove_empty_nodes() (*angr.analyses.decompiler.clinic.Clinic static method*), 407
remove_fakerets() (*angr.analyses.cfg.cfg_emulated.CFGEmulated static method*), 318
remove_false_branch() (*angr.rust.mixins.cfg_transformation_mixin.CFGTransformation method*), 1418
remove_hookset() (*angr.procedures.hookset.HookSet static method*), 1315
remove_instruction() (*angr.analyses.reassembler.Reassembler method*), 877
remove_jump_target() (*angr.rust.mixins.cfg_transformation_mixin.CFGTransformation method*), 1418
remove_labels() (*in module angr.analyses.decompiler.utils*), 723

remove_last_statement() (*in module angr.analyses.decompiler.utils*), 720
remove_last_statements() (*in module angr.analyses.decompiler.utils*), 720
remove_node() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 628
remove_node() (*angr.analyses.decompiler.structurer_nodes.SequenceNode method*), 711
remove_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1167
remove_node() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1183
remove_node() (*angr.utils.graph.DirectedGraphHelper method*), 1688
remove_node_and_graph_node() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1172
remove_obj() (*angr.analyses.cfg.cfb.CFBlanket method*), 306
remove_object() (*angr.keyed_region.KeyedRegion method*), 1118
remove_object() (*angr.keyed_region.RegionObject method*), 1116
remove_patch() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
remove_preconstraints() (*angr.state_plugins.preconstrainer.SimStatePreconstrainer method*), 1583
remove_redundant_jumps() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase static method*), 719
remove_technique() (*angr.sim_manager.SimulationManager method*), 1443
remove_types() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1288
remove_unnecessary_stuff() (*angr.analyses.reassembler.Reassembler method*), 877
remove_unnecessary_stuff_glibc() (*angr.analyses.reassembler.Reassembler method*), 878
remove_use() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1263
remove_uses() (*angr.knowledge_plugins.key_definitions.uses.Uses method*), 1263
remove_variable() (*angr.analyses.decompiler.optimization_passes.engine_base.SimplifierAILState method*), 575
remove_variable() (*angr.keyed_region.KeyedRegion method*), 1118
remove_variable_by_atom() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1284
remove_vvar_def() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 886
RemoveCascadingConversions (*class in angr.analyses.decompiler.peephole_optimizations.remove_cascad* 614
RemoveConstInsert (*class in angr.analyses.decompiler.peephole_optimizations.remove_const_* 614
RemoveCxxConstructorCalls (*class in angr.analyses.decompiler.peephole_optimizations.remove_cxx_de* 614
RemoveEmptyIfBody (*class in angr.analyses.decompiler.peephole_optimizations.remove_empty_* 614
RemoveNodeNotice, 398
RemoveNoopConversions (*class in angr.analyses.decompiler.peephole_optimizations.remove_noop_c* 615
RemoveRedundantBitmasks (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 615
RemoveRedundantConversions (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 615
RemoveRedundantDerefs (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 616
RemoveRedundantInsert (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 616
RemoveRedundantITEBranches (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 616
RemoveRedundantITEComparisons (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 616
RemoveRedundantNots (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 617
RemoveRedundantReinterprets (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 617
RemoveRedundantShifts (*class in angr.analyses.decompiler.peephole_optimizations.remove_reduna* 617
RemoveRedundantShiftsAroundComparators (*class in angr.analyses.decompiler.peephole_optimizations.remove_redu* 618
rename() (*angr.knowledge_plugins.types.TypesStore method*), 1280
rename() (*in module angr.mcp.edit_tools*), 1299
rename_function() (*in module angr.analyses.decompiler.edits.ops*), 442
rename_global() (*in module angr.analyses.decompiler.edits.ops*), 446
rename_variable() (*in module*

*angr.analyses.decompiler.edits.ops*), 443
render() (*angr.analyses.disassembly.Disassembly method*), 743
render() (*angr.analyses.disassembly.DisassemblyPiece method*), 739
render() (*angr.sim_type.SimTypeBitfield method*), 1474
render_notes() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 685
render_notes() (*angr.analyses.decompiler.structured_codegen.rust_ast.RustStructuredCodeGenerator method*), 708
render_text() (*angr.analyses.decompiler.structured_codegen.c.CStructuredCodeGenerator method*), 685
render_text() (*angr.analyses.decompiler.structured_codegen.rust_ast.RustStructuredCodeGenerator method*), 708
RenderResult (*in module angr.analyses.decompiler.structured_codegen.c*), 670
reopen_lmdb() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method*), 1278
reopen_lmdb_databases() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDb method*), 1278
reopen_rtdb() (*angr.knowledge_plugins.rtdb.rtdb.RuntimeDbForkContext method*), 1277
REPEATABLE (*angr.knowledge_plugins.CommentKind attribute*), 1124
REPEATABLE (*angr.knowledge_plugins.comments.CommentKind attribute*), 1189
repeatable_comments_at() (*angr.knowledge_plugins.comments.Comments method*), 1191
RepHook (*class in angr.exploration_techniques.tracer*), 1108
replace() (*angr.analyses.decompiler.structurer_nodes.IncompleteSwitchCaseHead method*), 714
replace() (*angr.analyses.s_propagator.SPropagator static method*), 880
replace() (*angr.analyses.typehoon.typeconsts.Array method*), 911
replace() (*angr.analyses.typehoon.typeconsts.Enum method*), 912
replace() (*angr.analyses.typehoon.typeconsts.Function method*), 912
replace() (*angr.analyses.typehoon.typeconsts.Pointer method*), 910
replace() (*angr.analyses.typehoon.typeconsts.RustEnum method*), 911
replace() (*angr.analyses.typehoon.typeconsts.TypeConstant method*), 907
replace() (*angr.analyses.typehoon.typevars.Add method*), 916
replace() (*angr.analyses.typehoon.typevars.DerivedTypeVariable method*), 918
replace() (*angr.analyses.typehoon.typevars.Existence method*), 915
replace() (*angr.analyses.typehoon.typevars.Sub method*), 916
replace() (*angr.analyses.typehoon.typevars.Subtype method*), 915
replace() (*angr.keyed_region.KeyedRegion method*), 1117
replace() (*angr.procedures.stubs.format_parser.FormatString method*), 1278
replace_all() (*angr.storage.memory_mixins.convenient_mappings_mixin method*), 1614
replace_all() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1648
replace_all() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin method*), 1663
replace_all_with_offsets() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 1655
replace_and_build() (*angr.analyses.decompiler.block_simplifier.BlockSimplifier static method*), 401
replace_argument_pairs() (*in module angr.rust.optimization_passes.utils*), 1431
replace_call() (*angr.rust.optimization_passes.macro.format_macro_simplifier method*), 1424
replace_call() (*angr.rust.optimization_passes.str_argument_simplifier.StrArgumentSimplifier method*), 1429
replace_jump_target() (*angr.rust.mixins.cfg_transformation_mixin.CFGTransformationMixin method*), 1418
replace_last_statement() (*in module angr.analyses.decompiler.utils*), 720
replace_node() (*angr.utils.graph.DirectedGraphHelper method*), 1688
replace_node_in_graph() (*in module angr.analyses.decompiler.optimization_passes.duplication_reverter*), 572
replace_node_in_node() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase static method*), 719
replace_nodes() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 630
replace_nodes() (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase method*), 719
replace_nodes() (*angr.utils.graph.DirectedGraphHelper method*), 1688
replace_nodes_both() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer method*), 718
replace_pointer_pts_to() (*in module angr.utils.types*), 1701
replace_stmt() (*angr.rust.mixins.cfa_mixin.CFAMixin static method*), 1417
replacements (*angr.analyses.propagator.propagator.PropagatorAnalysis*

property), 797
replacements (angr.analyses.s_propagator.SPropagator property), 879
replacements (angr.utils.balancer.Balancer property), 1673
report() (angr.exploration_techniques.suggestions.Suggestions static method), 1106
repr() (angr.rust.sim_type.RustSimEnum method), 1434
repr() (angr.rust.sim_type.RustSimStruct method), 1433
repr() (angr.rust.sim_type.RustSimType method), 1431
repr() (angr.rust.sim_type.RustSimTypeArray method), 1433
repr() (angr.rust.sim_type.RustSimTypeBottom method), 1434
repr() (angr.rust.sim_type.RustSimTypeFunction method), 1432
repr() (angr.rust.sim_type.RustSimTypeInt method), 1431
repr() (angr.rust.sim_type.RustSimTypeNumOffset method), 1433
repr() (angr.rust.sim_type.RustSimTypeOption method), 1435
repr() (angr.rust.sim_type.RustSimTypeReference method), 1432
repr() (angr.rust.sim_type.RustSimTypeResult method), 1435
repr() (angr.rust.sim_type.RustSimTypeSlice method), 1433
repr() (angr.rust.sim_type.RustSimTypeVec method), 1433
repr() (angr.rust.typehoon.translator.RustSimTypeTempRef method), 1436
repr_addr() (in module angr.codenode), 987
representative() (angr.analyses.decompiler.region_overseer.RegionOverseer method), 626
request_knowledge() (angr.knowledge_base.KnowledgeBase method), 1119
request_plugin() (angr.misc.plugins.PluginPreset method), 1320
require_cache() (in module angr.analyses.decompiler.edits.cache), 436
REQUIRE_CFG_STATES (angr.exploration_techniques.CallFunctionGoal attribute), 1087
REQUIRE_CFG_STATES (angr.exploration_techniques.director.BaseGoal attribute), 1099
REQUIRE_CFG_STATES (angr.exploration_techniques.director.CallFunctionGoal attribute), 1100
REQUIRE_DATA_C (angr.engines.pcode.lifter.Lifter attribute), 1041
REQUIRE_DATA_PY (angr.engines.pcode.lifter.Lifter attribute), 1041
requires_evidence (angr.analyses.decompiler.known_patterns.gating.AllOf property), 514
requires_evidence (angr.analyses.decompiler.known_patterns.gating.An property), 513
requires_evidence (angr.analyses.decompiler.known_patterns.gating.Co property), 513
requires_evidence (angr.analyses.decompiler.known_patterns.gating.Co property), 514
requires_evidence (angr.analyses.decompiler.known_patterns.gating.Pa property), 512
reraise() (angr.sim_manager.ErrorRecord method), 1448
reset() (angr.ailment.block_walker.AILBlockWalker method), 215
reset() (angr.ailment.Manager method), 212
reset() (angr.ailment.manager.Manager method), 224
reset() (angr.analyses.forward_analysis.visitors.graph.GraphVisitor method), 767
reset() (angr.state_plugins.unicorn_engine.Uniwrapper method), 1603
reset() (angr.utils.graph.DirectedGraphHelper method), 1688
reset_ident_counters() (angr.analyses.decompiler.structured_codegen.base.BaseStructur method), 670
reset_initial_regs() (angr.block.Block static method), 961
reset_memory_data_addrs() (angr.analyses.cfg.cfg_base.CFGBase method), 309
reset_prototype() (angr.analyses.reaching_definitions.function_handle method), 841
reset_tmp_uses_only() (angr.utils.ssa.combined_uses_collector.VVarAndTmpUsesCollec method), 1699
reset_uses() (angr.knowledge_plugins.key_definitions.live_definitions.Li method), 1251
resolvable (angr.state_plugins.debug_variables.SimDebugVariable property), 1546
resolvable (angr.state_plugins.view.SimMemView property), 1607
resolve() (angr.analyses.cfg.indirect_jump_resolvers.aarch64_macho_go method), 331
resolve() (angr.analyses.cfg.indirect_jump_resolvers.arm_elf_fast.ArmEl method), 332
resolve() (angr.analyses.cfg.indirect_jump_resolvers.const_resolver.Cons method), 332
resolve() (angr.analyses.cfg.indirect_jump_resolvers.jumptable.JumpTab method), 337
resolve() (angr.analyses.cfg.indirect_jump_resolvers.memload_resolver.M method), 338
resolve() (angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.MipsE method), 339
resolve() (angr.analyses.cfg.indirect_jump_resolvers.resolver.IndirectJum method), 340
resolve() (angr.analyses.decompiler.known_patterns.pattern.CppRef

*method*), 519
resolve() (*angr.analyses.decompiler.structured_codegen.c_serializable.SerializableCFunction method*), 690
resolve() (*angr.rust.optimization_passes.macro.format_macro_simplifier.FormatMacroSimplifier method*), 1424
resolve() (*angr.sim_type.SimTypeBitfield method*), 1474
resolve() (*angr.sim_type.SimTypeEnum method*), 1473
resolve_abstract_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
resolve_concrete_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
resolve_field() (*in module angr.engines.soot.field_dispatcher*), 1052
resolve_function() (*in module angr.analyses.decompiler.edits.resolve*), 447
resolve_invoke() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
resolve_method() (*in module angr.engines.soot.method_dispatcher*), 1053
resolve_name_collisions() (*angr.analyses.decompiler.semantic_naming.orchestrator.SemanticNamingOrchestrator method*), 650
resolve_pattern_selection() (*in module angr.analyses.decompiler.known_patterns.registry*), 525
resolve_peephole_optimizations() (*angr.analyses.decompiler.clinic.Clinic method*), 408
resolve_register() (*angr.state_plugins.light_registers.SimLightRegisters method*), 1577
resolve_simvar() (*angr.analyses.decompiler.structured_codegen.c_printer.CStructuredCodePrinter method*), 689
resolve_special_dispatch() (*angr.analyses.soot_class_hierarchy.SootClassHierarchy method*), 893
resolve_tags() (*angr.analyses.decompiler.structured_codegen.c_serializable.SerializableCFunction method*), 690
resolve_type() (*angr.analyses.decompiler.structured_codegen.c_printer.CStructuredCodePrinter method*), 689
resolve_typeref() (*in module angr.analyses.decompiler.known_patterns.pattern*), 519
resolve_variable() (*in module angr.analyses.decompiler.edits.resolve*), 448
resolved (*angr.state_plugins.debug_variables.SimDebugVariable property*), 1546
resolved (*angr.state_plugins.view.SimMemView property*), 1607
resolved_indirect_jumps (*angr.knowledge_base.KnowledgeBase property*), 1119
ResolvedVariable (*class in angr.analyses.decompiler.edits.resolve*), 447
resource_event() (*in module angr.state_plugins.debug_variables.DebugVariableSimInspectPlugin event*), 1587
restore_graph() (*angr.analyses.decompiler.optimization_passes.lowered_switch_simplifier.LoweredSwitchSimplifier static method*), 585
restore_user_edits() (*in module angr.analyses.decompiler.edits.cache*), 436
result_type() (*angr.engines.light.engine.IRTop method*), 1010
ResultType (*class in angr.analyses.purity.engine*), 802
RESUME (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Case2RegResolver attribute*), 338
resume() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 312
resume_with_new_graph() (*angr.analyses.forward_analysis.visitors.function_graph.FunctionGraphVisitor method*), 766
ResumeHook (*class in angr.analyses.cfg.cfg_fast*), 316
resymbolize() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 1596
ret() (*angr.sim_procedure.SimProcedure method*), 1451
ret() (*angr.state_plugins.callstack.CallStack method*), 1544
RET_ADDR (*angr.analyses.decompiler.stack_item.StackItemType attribute*), 659
ret_atoms (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
ret_atoms (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
ret_errno() (*angr.state_plugins.libc.SimStateLibc method*), 1576
ret_sites (*angr.knowledge_plugins.functions.function.Function property*), 1213
ret_values (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
ret_values (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
ret_values_deps (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
ret_values_deps (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
retaddr_popped (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
retaddr_popped (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 808
RetAddrSaveSimplifier (*class in angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier*), 591
RetExprRewriter (*class in angr.rust.optimization_passes.ret_expr_rewriter*), 1428
retout_sites (*angr.knowledge_plugins.functions.function.Function property*), 1213

retreg (*class in angr.procedures.testing.retreg*), 1380
retsite() (*angr.procedures.posix.pthread.pthread_once method*), 1374
Return (*angr.protos.primitives_pb2.Edge attribute*), 1403
RETURN (*angr.state_plugins.inspect.EventType attribute*), 1565
Return (*class in angr.ailment.statement*), 225
RETURN_ADDR (*angr.calling_conventions.SimCC attribute*), 967
return_addr (*angr.calling_conventions.SimCC property*), 969
RETURN_ADDR (*angr.calling_conventions.SimCCAArch64 attribute*), 978
RETURN_ADDR (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 978
RETURN_ADDR (*angr.calling_conventions.SimCCAArch64WindowsSyscall attribute*), 979
RETURN_ADDR (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 976
RETURN_ADDR (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 976
RETURN_ADDR (*angr.calling_conventions.SimCCARM attribute*), 977
RETURN_ADDR (*angr.calling_conventions.SimCCARMHF attribute*), 977
RETURN_ADDR (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 977
RETURN_ADDR (*angr.calling_conventions.SimCCARMWindowsSyscall attribute*), 978
RETURN_ADDR (*angr.calling_conventions.SimCCCdecl attribute*), 972
RETURN_ADDR (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
RETURN_ADDR (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
RETURN_ADDR (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 972
RETURN_ADDR (*angr.calling_conventions.SimCCN64 attribute*), 980
RETURN_ADDR (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 981
RETURN_ADDR (*angr.calling_conventions.SimCCO32 attribute*), 980
RETURN_ADDR (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 980
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC attribute*), 981
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC64 attribute*), 982
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 982
RETURN_ADDR (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 981
RETURN_ADDR (*angr.calling_conventions.SimCCRISCV64 attribute*), 979
RETURN_ADDR (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 979
RETURN_ADDR (*angr.calling_conventions.SimCCS390X attribute*), 983
RETURN_ADDR (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 983
RETURN_ADDR (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
RETURN_ADDR (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 974
RETURN_ADDR (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 974
RETURN_ADDR (*angr.SimCC attribute*), 173
return_from_invocation() (*angr.procedures.java_jni.method_calls.CallMethod method*), 1342
return_from_invocation() (*angr.procedures.java_jni.method_calls.CallMethodA method*), 1342
return_from_invocation() (*angr.procedures.java_jni.method_calls.CallNonvirtualMethod method*), 1343
return_from_invocation() (*angr.procedures.java_jni.method_calls.CallNonvirtualMethodA method*), 1343
return_from_invocation() (*angr.procedures.java_jni.method_calls.CallStaticMethod method*), 1345
return_from_invocation() (*angr.procedures.java_jni.method_calls.CallStaticMethodA method*), 1345
return_from_invocation() (*angr.procedures.java_jni.object_operations.NewObject method*), 1347
return_in_implicit_outparam() (*angr.calling_conventions.SimCC method*), 969
return_in_implicit_outparam() (*angr.calling_conventions.SimCCCdecl method*), 972
return_in_implicit_outparam() (*angr.calling_conventions.SimCCMicrosoftAMD64 method*), 973
return_in_implicit_outparam() (*angr.calling_conventions.SimCCMicrosoftFastcall method*), 972
return_in_implicit_outparam() (*angr.calling_conventions.SimCCSystemVAMD64 method*), 975
return_ty (*angr.procedures.java_jni.array_operations.GetArrayElements attribute*), 1339
return_ty (*angr.procedures.java_jni.array_operations.GetArrayLength*

*attribute*), 1338
return_ty (*angr.procedures.java_jni.array_operations.GetArrayRegion attribute*), 1339
return_ty (*angr.procedures.java_jni.array_operations.GetObjectArrayElement attribute*), 1339
return_ty (*angr.procedures.java_jni.array_operations.NewArray attribute*), 1338
return_ty (*angr.procedures.java_jni.array_operations.NewObjectArray attribute*), 1339
return_ty (*angr.procedures.java_jni.array_operations.ReleaseArrayElements attribute*), 1339
return_ty (*angr.procedures.java_jni.array_operations.SetArrayRegion attribute*), 1339
return_ty (*angr.procedures.java_jni.array_operations.SetObjectArrayElement attribute*), 1339
return_ty (*angr.procedures.java_jni.class_and_interface_operations.FindClass attribute*), 1340
return_ty (*angr.procedures.java_jni.class_and_interface_operations.GetSuperclass attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetBooleanField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.GetByteField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.GetCharField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.GetFieldID attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetIntField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.GetLongField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.GetObjectField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.GetShortField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.GetStaticFieldID attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetStaticObjectField attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetStaticBooleanField attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetStaticByteField attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetStaticCharField attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetStaticIntField attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetStaticLongField attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.GetStaticShortField attribute*), 1340
return_ty (*angr.procedures.java_jni.field_access.SetField attribute*), 1341
return_ty (*angr.procedures.java_jni.field_access.SetStaticField attribute*), 1340
return_ty (*angr.procedures.java_jni.global_and_local_refs.DeleteGlobalRef attribute*), 1341
return_ty (*angr.procedures.java_jni.global_and_local_refs.DeleteLocalRef attribute*), 1342
return_ty (*angr.procedures.java_jni.global_and_local_refs.NewGlobalRef attribute*), 1341
return_ty (*angr.procedures.java_jni.global_and_local_refs.NewLocalRef attribute*), 1341
return_ty (*angr.procedures.java_jni.JNISimProcedure attribute*), 1338
return_ty (*angr.procedures.java_jni.method_calls.CallBooleanMethod attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.CallBooleanMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallByteMethod attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.CallByteMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallCharMethod attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.CallCharMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallIntMethod attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.CallIntMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallLongMethod attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallLongMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallMethodBase attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualBooleanMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualBooleanMethodA attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualByteMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualByteMethodA attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualCharMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualCharMethodA attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualIntMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualIntMethodA attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualLongMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualLongMethodA attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualObjectMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualObjectMethodA*

*attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualShortMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualShortMethodA attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualVoidMethod attribute*), 1344
return_ty (*angr.procedures.java_jni.method_calls.CallNonvirtualVoidMethodA attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallObjectMethod attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.CallObjectMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallShortMethod attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.CallShortMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallStaticBooleanMethod attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallStaticBooleanMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticByteMethod attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallStaticByteMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticCharMethod attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallStaticCharMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticDoubleMethod attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticDoubleMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticFloatMethod attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallStaticFloatMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticIntMethod attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallStaticIntMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticLongMethod attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticLongMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticObjectMethod attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallStaticObjectMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticShortMethod attribute*), 1345
return_ty (*angr.procedures.java_jni.method_calls.CallStaticShortMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticVoidMethod attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallStaticVoidMethodA attribute*), 1346
return_ty (*angr.procedures.java_jni.method_calls.CallVoidMethod attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.CallVoidMethodA attribute*), 1343
return_ty (*angr.procedures.java_jni.method_calls.GetMethodID attribute*), 1342
return_ty (*angr.procedures.java_jni.method_calls.GetStaticMethodID attribute*), 1342
return_ty (*angr.procedures.java_jni.not_implemented.UnsupportedJNIFunction attribute*), 1346
return_ty (*angr.procedures.java_jni.object_operations.AllocObject attribute*), 1347
return_ty (*angr.procedures.java_jni.object_operations.GetObjectClass attribute*), 1347
return_ty (*angr.procedures.java_jni.object_operations.IsInstanceOf attribute*), 1347
return_ty (*angr.procedures.java_jni.object_operations.IsSameObject attribute*), 1347
return_ty (*angr.procedures.java_jni.object_operations.NewObject attribute*), 1347
return_ty (*angr.procedures.java_jni.string_operations.GetStringUTFChars attribute*), 1347
return_ty (*angr.procedures.java_jni.string_operations.GetStringUTFLength attribute*), 1347
return_ty (*angr.procedures.java_jni.string_operations.NewStringUTF attribute*), 1347
return_ty (*angr.procedures.java_jni.string_operations.ReleaseStringUTFChars attribute*), 1347
return_ty (*angr.procedures.java_jni.version_information.GetVersion attribute*), 1347
return_type (*angr.sim_procedure.SimProcedure property*), 1452
RETURN_VAL (*angr.calling_conventions.SimCC attribute*), 967
RETURN_VAL (*angr.calling_conventions.SimCCAArch64 attribute*), 978
RETURN_VAL (*angr.calling_conventions.SimCCAArch64LinuxSyscall attribute*), 978
RETURN_VAL (*angr.calling_conventions.SimCCAArch64WindowsSyscall attribute*), 979
RETURN_VAL (*angr.calling_conventions.SimCCAMD64LinuxSyscall attribute*), 976
RETURN_VAL (*angr.calling_conventions.SimCCAMD64WindowsSyscall attribute*), 976
RETURN_VAL (*angr.calling_conventions.SimCCARM attribute*), 977
RETURN_VAL (*angr.calling_conventions.SimCCARMHF attribute*), 977
RETURN_VAL (*angr.calling_conventions.SimCCARMLinuxSyscall attribute*), 977
RETURN_VAL (*angr.calling_conventions.SimCCARMWindowsSyscall attribute*), 978
RETURN_VAL (*angr.calling_conventions.SimCCCdecl attribute*), 971
RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
RETURN_VAL (*angr.calling_conventions.SimCCGoAMD64ABI0 attribute*), 976
RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
RETURN_VAL (*angr.calling_conventions.SimCCMicrosoftFastcall*

*attribute*), 972
RETURN_VAL (*angr.calling_conventions.SimCCN64 attribute*), 980
RETURN_VAL (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 981
RETURN_VAL (*angr.calling_conventions.SimCCO32 attribute*), 980
RETURN_VAL (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 980
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC attribute*), 981
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC64 attribute*), 982
RETURN_VAL (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 982
RETURN_VAL (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 981
RETURN_VAL (*angr.calling_conventions.SimCCRISCV64 attribute*), 979
RETURN_VAL (*angr.calling_conventions.SimCCRISCV64LinuxSyscall attribute*), 979
RETURN_VAL (*angr.calling_conventions.SimCCS390X attribute*), 983
RETURN_VAL (*angr.calling_conventions.SimCCS390XLinuxSyscall attribute*), 983
RETURN_VAL (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
RETURN_VAL (*angr.calling_conventions.SimCCX86LinuxSyscall attribute*), 974
RETURN_VAL (*angr.calling_conventions.SimCCX86WindowsSyscall attribute*), 974
RETURN_VAL (*angr.SimCC attribute*), 173
return_val() (*angr.calling_conventions.SimCC method*), 969
ReturnChar (*class in angr.procedures.stubs.ReturnChar*), 1377
ReturnDeduplicator (*class in angr.analyses.decompiler.optimization_passes.ret_deduplicator*), 591
ReturnDuplicatorBase (*class in angr.analyses.decompiler.optimization_passes.return_duplicator_base*), 592
ReturnDuplicatorHigh (*class in angr.analyses.decompiler.optimization_passes.return_duplicator_high*), 592
ReturnDuplicatorLow (*class in angr.analyses.decompiler.optimization_passes.return_duplicator_low*), 593
returning (*angr.knowledge_plugins.functions.function.Function property*), 1211
ReturnMaker (*class in angr.analyses.decompiler.return_maker*), 642
returnty (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 457
returnty (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521
returnty (*angr.analyses.decompiler.KnownPattern attribute*), 379
returnty() (*angr.analyses.decompiler.variable_map.VariableMap method*), 729
returnty_factory (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 457
returnty_factory (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521
returnty_factory (*angr.analyses.decompiler.KnownPattern attribute*), 379
ReturnUnconstrained (*class in angr.procedures.stubs.ReturnUnconstrained*), 1377
ReturnValueTag (*class in angr.knowledge_plugins.key_definitions.tag*), 1261
reverse_post_order_sort_nodes() (*angr.utils.graph.GraphUtils static method*), 1686
reverse_view() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 632
revisit_node() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 768
revisit_successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisitor method*), 768
rewind (*class in angr.procedures.libc.rewind*), 1359
rewrite_phi_sources() (*in module angr.analyses.decompiler.known_patterns.block_split*), 486
RewriteBitExtractions (*class in angr.analyses.decompiler.peephole_optimizations.rewrite_bit_ext*), 618
RewriteConvMul (*class in angr.analyses.decompiler.peephole_optimizations.rewrite_conv_m*), 618
RewriteCxxOperatorCalls (*class in angr.analyses.decompiler.peephole_optimizations.rewrite_cxx_op*), 619
RewriteMipsGpLoads (*class in angr.analyses.decompiler.peephole_optimizations.rewrite_mips_g*), 619
RewriteStdStringCallWalker (*class in angr.analyses.decompiler.optimization_passes.eager_std_string_e*), 574
RewritingAnalysis (*class in angr.analyses.decompiler.ssailification.rewriting*), 653
RewritingState (*class in angr.analyses.decompiler.ssailification.rewriting_state*), 654

RichR (*class in angr.analyses.variable_recovery.engine_base*), 924
RIGHT (*angr.analyses.typehoon.simple_solver.ConstraintGraphTag attribute*), 901
riscv64g_calculate_fflags_feq_d() (*in module angr.engines.vex.claripy.ccall*), 1073
riscv64g_calculate_fflags_feq_s() (*in module angr.engines.vex.claripy.ccall*), 1073
riscv64g_calculate_fflags_fle_d() (*in module angr.engines.vex.claripy.ccall*), 1073
riscv64g_calculate_fflags_fle_s() (*in module angr.engines.vex.claripy.ccall*), 1073
riscv64g_calculate_fflags_flt_d() (*in module angr.engines.vex.claripy.ccall*), 1073
riscv64g_calculate_fflags_flt_s() (*in module angr.engines.vex.claripy.ccall*), 1073
rollback() (*angr.analyses.decompiler.region_overlay.OverlayManager method*), 624
RolRorRewriter (*class in angr.analyses.decompiler.peephole_optimizations.rol_ror*), 619
root (*angr.state_plugins.filesystem.SimFilesystem property*), 1549
RShift (*angr.engines.light.ArithmeticExpression attribute*), 1006
RShift (*angr.engines.light.data.ArithmeticExpression attribute*), 1009
rt_sigaction (*class in angr.procedures.linux_kernel.sigaction*), 1367
run() (*angr.analyses.congruency_check.CongruencyCheck method*), 354
run() (*angr.analyses.identifier.identify.Identifier method*), 785
run() (*angr.distributed.server.Server method*), 994
run() (*angr.distributed.worker.Worker method*), 995
run() (*angr.emulator.Emulator method*), 995
run() (*angr.exploration_techniques.tracer.RepHook method*), 1108
run() (*angr.procedures.posix.inet_ntoa.inet_ntoa method*), 1372
run() (*angr.sim_manager.SimulationManager method*), 1443
run() (*angr.sim_procedure.SimProcedure method*), 1450
run_initializer() (*angr.procedures.linux_loader.sim_loader.LinuxLoader method*), 1368
run_pelf() (*in module angr.flirt.build_sig*), 1115
run_sigmake() (*in module angr.flirt.build_sig*), 1115
run_test() (*angr.analyses.identifier.functions.recv_until.receive_until method*), 778
run_test() (*angr.analyses.identifier.functions.recv_until.recv_until method*), 777
Runner (*class in angr.analyses.identifier.runner*), 785
RuntimeDb (*class in angr.knowledge_plugins.rtdb.rtdb*), 1277
RuntimeDbForkCondom (*class in angr.knowledge_plugins.rtdb.rtdb*), 1277
runtimes (*angr.analyses.decompiler.known_patterns.KnownPatternTempla attribute*), 463
runtimes (*angr.analyses.decompiler.known_patterns.templates.KnownPatt attribute*), 532
RustAILBlock (*class in angr.analyses.decompiler.structured_codegen.rust*), 693
RustArray (*class in angr.analyses.decompiler.structured_codegen.rust*), 700
RustArrayTypeLength (*class in angr.analyses.decompiler.structured_codegen.rust*), 706
RustAssignment (*class in angr.analyses.decompiler.structured_codegen.rust*), 697
RustBinaryOp (*class in angr.analyses.decompiler.structured_codegen.rust*), 702
RustBreak (*class in angr.analyses.decompiler.structured_codegen.rust*), 695
RustCallingConvention (*class in angr.rust.optimization_passes.rust_calling_convention*), 1428
RustCallingConventionAnalysis (*class in angr.rust.analyses.rust_calling_convention.rust_calling_conventi* 1409
RustCallingConventionModel (*class in angr.rust.analyses.rust_calling_convention.rust_calling_conventi* 1410
RustCallingConventions (*class in angr.rust.knowledge_plugins.rust_calling_conventions*), 1413
RustClosingObject (*class in angr.analyses.decompiler.structured_codegen.rust*), 706
RustConstant (*class in angr.analyses.decompiler.structured_codegen.rust*), 703
RustConstruct (*class in angr.analyses.decompiler.structured_codegen.rust*), 692
RustControlFlowRecoverer (*class in angr.analyses.decompiler.structured_codegen.rust*), 696
RustcVersionIdentification (*class in angr.rust.analyses.rustc_version_identification*), 1410
RustDirtyExpression (*class in angr.analyses.decompiler.structured_codegen.rust*), 706
RustDirtyStatement (*class in*

*angr.analyses.decompiler.structured_codegen.rust*), 699
RustDoWhileLoop (*class in angr.analyses.decompiler.structured_codegen.rust*), 694
RustEnum (*class in angr.ailment.expression*), 222
RustEnum (*class in angr.analyses.decompiler.structured_codegen.rust*), 700
RustEnum (*class in angr.analyses.typehoon.typeconsts*), 911
RustExpression (*class in angr.analyses.decompiler.structured_codegen.rust*), 693
RustFakeVariable (*class in angr.analyses.decompiler.structured_codegen.rust*), 701
RustForLoop (*class in angr.analyses.decompiler.structured_codegen.rust*), 694
RustFunction (*class in angr.analyses.decompiler.structured_codegen.rust*), 692
RustFunctionCall (*class in angr.analyses.decompiler.structured_codegen.rust*), 697
RustFunctionLikeMacro (*class in angr.analyses.decompiler.structured_codegen.rust*), 697
RustGoto (*class in angr.analyses.decompiler.structured_codegen.rust*), 698
RustIfBreak (*class in angr.analyses.decompiler.structured_codegen.rust*), 695
RustIfElse (*class in angr.analyses.decompiler.structured_codegen.rust*), 695
RustIfLet (*class in angr.analyses.decompiler.structured_codegen.rust*), 696
RustIncompleteSwitchCase (*class in angr.analyses.decompiler.structured_codegen.rust*), 699
RustIndexedVariable (*class in angr.analyses.decompiler.structured_codegen.rust*), 702
RustInfiniteLoop (*class in angr.analyses.decompiler.structured_codegen.rust*), 694
RustITE (*class in angr.analyses.decompiler.structured_codegen.rust*), 704
RustLabel (*class in angr.analyses.decompiler.structured_codegen.rust*), 699
RustLet (*class in angr.analyses.decompiler.structured_codegen.rust*), 700
RustLoop (*class in angr.analyses.decompiler.structured_codegen.rust*), 694
RustMultiStatementExpression (*class in angr.analyses.decompiler.structured_codegen.rust*), 705
RustPatternMatch (*class in angr.analyses.decompiler.structured_codegen.rust*), 696
RustRegister (*class in angr.analyses.decompiler.structured_codegen.rust*), 704
RustReturn (*class in angr.analyses.decompiler.structured_codegen.rust*), 698
RustSimEnum (*class in angr.rust.sim_type*), 1434
RustSimStruct (*class in angr.rust.sim_type*), 1433
RustSimType (*class in angr.rust.sim_type*), 1431
RustSimTypeArray (*class in angr.rust.sim_type*), 1432
RustSimTypeBottom (*class in angr.rust.sim_type*), 1434
RustSimTypeFunction (*class in angr.rust.sim_type*), 1431
RustSimTypeInt (*class in angr.rust.sim_type*), 1431
RustSimTypeNumOffset (*class in angr.rust.sim_type*), 1433
RustSimTypeOption (*class in angr.rust.sim_type*), 1435
RustSimTypeReference (*class in angr.rust.sim_type*), 1432
RustSimTypeResult (*class in angr.rust.sim_type*), 1435
RustSimTypeSize (*class in angr.rust.sim_type*), 1431
RustSimTypeSlice (*class in angr.rust.sim_type*), 1433
RustSimTypeStrRef (*class in angr.rust.sim_type*), 1435
RustSimTypeTempRef (*class in angr.rust.typehoon.translator*), 1436
RustSimTypeUnit (*class in angr.rust.sim_type*), 1435
RustSimTypeVec (*class in angr.rust.sim_type*), 1433
RustStatement (*class in angr.analyses.decompiler.structured_codegen.rust*), 693
RustStatements (*class in angr.analyses.decompiler.structured_codegen.rust*), 693
RustStringLiteral (*class in angr.analyses.decompiler.structured_codegen.rust*), 701
RustStruct (*class in angr.analyses.decompiler.structured_codegen.rust*), 699
RustStructField (*class in angr.analyses.decompiler.structured_codegen.rust*), 700
RustStructFieldNameDef (*class in angr.analyses.decompiler.structured_codegen.rust*), 706
RustStructuredCodeGenerator (*class in angr.analyses.decompiler.structured_codegen.rust*), 706
RustStructuredCodeWalker (*class in angr.analyses.decompiler.structured_codegen.rust*),

708
RustSwitchCase (*class in angr.analyses.decompiler.structured_codegen.rust*), 696
RustSymbolRecovery (*class in angr.rust.analyses.rust_symbol_recovery*), 1410
RustTypeCast (*class in angr.analyses.decompiler.structured_codegen.rust*), 703
RustTypeHintsAnalysis (*class in angr.rust.analyses.rust_type_hints*), 1410
RustTypehoon (*class in angr.rust.typehoon.typehoon*), 1437
RustTypeTranslator (*class in angr.rust.typehoon.translator*), 1436
RustUnaryOp (*class in angr.analyses.decompiler.structured_codegen.rust*), 702
RustUnsupportedStatement (*class in angr.analyses.decompiler.structured_codegen.rust*), 698
RustVariable (*class in angr.analyses.decompiler.structured_codegen.rust*), 701
RustVariableField (*class in angr.analyses.decompiler.structured_codegen.rust*), 702
RustVectorConvert (*class in angr.analyses.decompiler.structured_codegen.rust*), 705
RustVEXCCallExpression (*class in angr.analyses.decompiler.structured_codegen.rust*), 705
RustWhileLoop (*class in angr.analyses.decompiler.structured_codegen.rust*), 694

# S

s2u() (*in module angr.utils.bits*), 1674
saffinity_mask (*angr.procedures.win32.GetProcessAffinityMask.GetProcessAffinityMask attribute*), 1383
SAILRSemanticError, 571
SAILRStructurer (*class in angr.analyses.decompiler.structuring.sailr*), 718
SarToSignedDiv (*class in angr.analyses.decompiler.peephole_optimizations.sar_to_signed_div*), 620
SAT (*angr.analyses.pathfinder.SuccessorsKind attribute*), 794
satisfiable() (*angr.sim_state.SimState method*), 1454
satisfiable() (*angr.state_plugins.solver.SimSolver method*), 1591
save_info() (*angr.angrdb.db.AngrDB static method*), 940
SAVED_BP (*angr.analyses.decompiler.stack_item.StackItemType attribute*), 659
SAVED_REGS (*angr.analyses.decompiler.stack_item.StackItemType attribute*), 659
scanf (*class in angr.procedures.libc.scanf*), 1359
SCANF_DELIMITERS (*angr.procedures.stubs.format_parser.FormatString attribute*), 1378
ScanfFormatParser (*class in angr.procedures.stubs.format_parser*), 1380
ScannerNextLine (*class in angr.procedures.java_util.scanner_nextline*), 1351
SCCPlaceholder (*class in angr.utils.graph*), 1685
ScopeOpsAnalyzer (*class in angr.analyses.deobfuscator.scope_ops_analyzer*), 738
ScopeOpsWalker (*class in angr.analyses.deobfuscator.scope_ops_analyzer*), 738
SDiv() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1586
search() (*angr.rustylib.SegmentList method*), 1439
section_alignment() (*angr.analyses.reassembler.Reassembler method*), 875
SecurityCheckRemover (*class in angr.rust.optimization_passes.security_check_remover*), 1428
SecurityCookieInit (*class in angr.simos.windows*), 1492
seek() (*angr.storage.file.SimFileDescriptorBase method*), 1614
seekable (*angr.SimFileBase attribute*), 179
seekable (*angr.storage.file.SimFileBase attribute*), 1610
seekable (*angr.storage.file.SimPacketsSlots attribute*), 1616
SegfaultError, 1603
Segment (*class in angr.storage.memory_mixins.regioned_memory.region_m*
1661
SegmentBoundary (*angr.knowledge_plugins.cfg.memory_data.MemoryDa attribute*), 1177
SegmentBoundary (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
SegmentBoundary (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
segments (*angr.storage.memory_mixins.regioned_memory.region_meta_m property*), 1662
select (*class in angr.procedures.posix.select*), 1374
selector() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1097
selector() (*angr.sim_manager.SimulationManager method*), 1444

SelfModifyingCodeAnalysis (*class in angr.analyses.smc*), 892
SEMANTIC_VARIABLE_NAMING (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
SemanticNamingBase (*class in angr.analyses.decompiler.semantic_naming.naming_base*), 648
SemanticNamingOrchestrator (*class in angr.analyses.decompiler.semantic_naming.orchestrator*), 650
send (*class in angr.procedures.posix.send*), 1374
SeqNodeDephication (*class in angr.analyses.decompiler.dephication.seqnode_dephication*), 420
SeqNodeRewriter (*class in angr.analyses.decompiler.dephication.seqnode_dephication*), 420
sequence_matcher_similarity() (*angr.exploration_techniques.unique.UniqueSearch static method*), 1109
sequence_to_blocks() (*in module angr.analyses.decompiler.utils*), 726
sequence_to_statements() (*in module angr.analyses.decompiler.utils*), 726
SequenceNode (*class in angr.analyses.decompiler.structurer_nodes*), 711
SequenceOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 588
SequenceToBlocks (*class in angr.analyses.decompiler.seq_to_blocks*), 652
SequenceWalker (*class in angr.analyses.decompiler.sequence_walker*), 652
Serializable (*class in angr.serializable*), 1440
SerializableCounter (*class in angr.calling_conventions*), 963
SerializableIterator (*class in angr.calling_conventions*), 963
SerializableListIterator (*class in angr.calling_conventions*), 963
serialize() (*angr.analyses.decompiler.structured_codegen.c_utils.CExpr method*), 689
serialize() (*angr.knowledge_plugins.functions.function_parser.FunctionParser static method*), 1227
serialize() (*angr.serializable.Serializable method*), 1440
serialize_basic_block() (*in module angr.mcp.serializers*), 1302
serialize_cfg_stats() (*in module angr.mcp.serializers*), 1302
serialize_codegen() (*in module angr.analyses.decompiler.structured_codegen.c_serialize*), 690
serialize_function() (*in module angr.mcp.serializers*), 1301
serialize_function_summary() (*in module angr.mcp.serializers*), 1301
serialize_subtree() (*in module angr.analyses.decompiler.structured_codegen.c_serialize*), 690
serialize_symbol() (*in module angr.mcp.serializers*), 1302
serialize_to_cmessage() (*angr.knowledge_plugins.cfg.indirect_jump.JumptableInfo method*), 1176
serialize_to_cmessage() (*angr.serializable.Serializable method*), 1440
serialize_xref() (*in module angr.mcp.serializers*), 1302
SerializeContext (*class in angr.analyses.decompiler.structured_codegen.c_serialize*), 688
Server (*class in angr.distributed.server*), 994
session_scope() (*angr.angrdb.db.AngrDB static method*), 940
SessionManager (*class in angr.mcp.session*), 1308
set() (*angr.knowledge_plugins.key_definitions.environment.Environment method*), 1248
set() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap_page.SymbolicBitmapPage method*), 1653
set_abi_cc() (*angr.procedures.definitions.SimSyscallLibrary method*), 1330
set_alloc_depth() (*angr.state_plugins.uc_manager.SimUCManager method*), 1598
set_args() (*angr.sim_procedure.SimProcedure method*), 1451
set_base_state() (*angr.analyses.identifier.custom_callable.IdentifierCallable method*), 770
set_base_state() (*angr.callable.Callable method*), 962
set_bck_chunk() (*angr.state_plugins.heap.heap_freelist.Chunk method*), 1557
set_brk() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
set_c_prototype() (*angr.procedures.definitions.SimLibrary method*), 1326
set_calling_convention() (*angr.analyses.decompiler.variable_map.VariableMap method*), 730
set_codegen() (*angr.analyses.decompiler.structured_codegen.c_serialize.SerializeContext method*), 690
set_comment() (*in module angr.analyses.decompiler.edits.ops*), 445
set_comments() (*in module angr.mcp.edit_tools*), 1300

set_custom_string() (angr.analyses.decompiler.variable_map.VariableMap method), 729
set_data() (angr.state_plugins.gdb.GDB method), 1551
set_default_cc() (angr.procedures.definitions.SimLibrary method), 1325
set_edge_eviction_enabled() (angr.knowledge_plugins.cfg.spilling_digraph.SpillingDiGraph method), 1188
set_entry_register_values() (angr.simos.linux.SimLinux method), 1488
set_fd_data() (angr.exploration_techniques.tracer.Tracer method), 1108
set_field() (angr.engines.soot.values.thisref.SimSootValue_ThisRef method), 1062
set_from_signature() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 1226
set_func_block_count() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 1225
set_function_prototype() (in module angr.analyses.decompiler.edits.ops), 444
set_function_returning() (angr.knowledge_plugins.functions.function_manager.FunctionManager method), 1224
set_fwd_chunk() (angr.state_plugins.heap.heap_freelist.Chunk method), 1557
set_global_type() (in module angr.analyses.decompiler.edits.ops), 446
set_graph() (angr.codenode.CodeNode method), 987
set_heap() (angr.state_plugins.gdb.GDB method), 1551
set_initial_regs() (angr.block.Block method), 961
set_kb() (angr.knowledge_plugins.plugin.KnowledgeBasePlugin method), 1266
set_kind() (angr.knowledge_plugins.comments.Comments method), 1190
set_last_block_details() (angr.state_plugins.unicorn_engine.Unicorn method), 1604
set_last_statement() (angr.annocfg.AnnotatedCFG method), 957
set_library_names() (angr.procedures.definitions.SimLibrary method), 1325
set_live_variables() (angr.knowledge_plugins.variables.variable_manager.VariableManager method), 1284
set_manager() (angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method), 1283
set_mode() (angr.sim_state.SimState method), 1455
set_names() (angr.procedures.definitions.SimTypeCollection method), 1324
set_non_returning() (angr.procedures.definitions.SimLibrary method), 1326
set_object() (angr.keyed_region.KeyedRegion method), 1117
set_object() (angr.keyed_region.RegionObject method), 1116
set_prev_freeness() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 1559
set_prototype() (angr.analyses.decompiler.variable_map.VariableMap method), 730
set_prototype() (angr.knowledge_plugins.callsite_prototypes.CallsitePr method), 1150
set_prototype() (angr.procedures.definitions.SimLibrary method), 1326
set_prototype() (angr.procedures.definitions.SimSyscallLibrary method), 1330
set_prototypes() (angr.procedures.definitions.SimLibrary method), 1326
set_prototypes() (angr.procedures.definitions.SimSyscallLibrary method), 1331
set_range() (angr.storage.memory_mixins.paged_memory.pages.symboli method), 1653
set_reference_values() (angr.analyses.decompiler.variable_map.VariableMap method), 729
set_reference_variable() (angr.analyses.decompiler.variable_map.VariableMap method), 730
set_regs() (angr.state_plugins.gdb.GDB method), 1551
set_regs() (angr.state_plugins.unicorn_engine.Unicorn method), 1604
set_return_val() (angr.calling_conventions.SimCC method), 969
set_return_val() (angr.calling_conventions.SimCCSyscall method), 974
set_returnty() (angr.analyses.decompiler.variable_map.VariableMap method), 730
set_rtdb() (angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method), 1187
set_rtdb() (angr.knowledge_plugins.cfg.spilling_digraph.SpillingDiGrap method), 1188
set_simgr() (angr.analyses.congruency_check.CongruencyCheck method), 354
set_size() (angr.sim_type.SimTypeRef method), 1475
set_size() (angr.state_plugins.heap.heap_freelist.Chunk method), 1557
set_size() (angr.state_plugins.heap.heap_ptmalloc.PTChunk method), 1559
set_stack() (angr.state_plugins.gdb.GDB method), 1551

set_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.region_mixin.RegionedMemoryMixin method*), 1663
set_stack_size() (*angr.storage.memory_mixins.regioned_memory.region_mixin.RegionedMemoryMixin method*), 1664
set_state() (*angr.state_plugins.plugin.SimStatePlugin method*), 1578
set_state_options() (*angr.analyses.congruency_check.CongruencyCheck method*), 354
set_states() (*angr.analyses.congruency_check.CongruencyCheck method*), 354
set_stops() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
set_symbolization_for_all_pages() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 1596
set_symbolized_target_range() (*angr.state_plugins.symbolizer.SimSymbolizer method*), 1596
set_tid_address (*class in angr.procedures.linux_kernel.set_tid_address*), 1367
set_tracking() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
set_tyenv() (*angr.state_plugins.scratch.SimStateScratch method*), 1583
set_type() (*angr.analyses.decompiler.structured_codegen.c.CExpression method*), 674
set_type() (*angr.analyses.decompiler.structured_codegen.rust.RustExpression method*), 693
set_type() (*in module angr.mcp.edit_tools*), 1299
set_unified_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1288
set_value() (*angr.calling_conventions.SimArrayArg method*), 966
set_value() (*angr.calling_conventions.SimComboArg method*), 965
set_value() (*angr.calling_conventions.SimFunctionArgument method*), 964
set_value() (*angr.calling_conventions.SimLyingRegArg method*), 971
set_value() (*angr.calling_conventions.SimReferenceArgument method*), 966
set_value() (*angr.calling_conventions.SimRegArg method*), 964
set_value() (*angr.calling_conventions.SimStackArg method*), 965
set_value() (*angr.calling_conventions.SimStructArg method*), 966
set_variable() (*angr.analyses.decompiler.variable_map.VariableMap method*), 729
set_variable() (*angr.keyed_region.KeyedRegion method*), 1117
set_variable() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1283
set_variable() (*angr.storage.memory_mixin.RegionedMemoryMixin method*), 1663
set_variable() (*angr.analyses.decompiler.variable_map.VariableMap method*), 729
set_variable_type() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1287
set_variable_type() (*in module angr.analyses.decompiler.edits.ops*), 444
set_variant() (*angr.analyses.decompiler.variable_map.VariableMap method*), 730
setall() (*angr.misc.loggers.Loggers static method*), 1316
SetArrayRegion (*class in angr.procedures.java_jni.array_operations*), 1339
setbuf (*class in angr.procedures.libc.setbuf*), 1359
SetField (*class in angr.procedures.java_jni.field_access*), 1341
SetObjectArrayElement (*class in angr.procedures.java_jni.array_operations*), 1339
setsockopt (*class in angr.procedures.posix.setsockopt*), 1375
setstate() (*angr.calling_conventions.ArgSession method*), 966
setstate() (*angr.calling_conventions.SerializableCounter method*), 963
setstate() (*angr.calling_conventions.SerializableIterator method*), 963
setstate() (*angr.calling_conventions.SerializableListIterator method*), 963
setstate() (*angr.calling_conventions.SimCC.ArgSession method*), 968
setstate() (*angr.calling_conventions.UsercallArgSession method*), 967
SetStaticField (*class in angr.procedures.java_jni.field_access*), 1340
SETTER (*angr.procedures.win32.local_storage.FlsFree attribute*), 1385
SETTER (*angr.procedures.win32.local_storage.TlsFree attribute*), 1385
setup() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1097
setup() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
setup_arguments() (*angr.engines.soot.engine.SootMixin static method*), 1047
setup_callsite() (*angr.calling_conventions.SimCC method*), 969
setup_callsite() (*angr.engines.soot.engine.SootMixin class method*), 1047

setup_flags() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
setup_gdt() (*angr.simos.simos.SimOS method*), 1490
setup_gdt() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
setup_state() (*angr.analyses.identifier.runner.Runner method*), 785
setup_terminal() (*in module angr.utils.formatting*), 1679
setvbuf (*class in angr.procedures.libc.setvbuf*), 1359
shallow_reverse() (*in module angr.utils.graph*), 1682
shared_common_conditional_dom() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.duplication_reverter.DuplicationReverter method*), 571
SHIFT_ELEMENT_SIZES (*in module angr.procedures.definitions.types_stl*), 1335
ShiftLeft (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransformationTypes attribute*), 334
ShiftRight (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransformationTypes attribute*), 334
ShlToMul (*class in angr.analyses.decompiler.peephole_optimizations.shl_to_mul*), 620
short_name (*angr.knowledge_plugins.functions.function.Function property*), 1217
short_repr (*angr.analyses.ddg.ProgramVariable property*), 362
short_repr (*angr.code_location.CodeLocation property*), 985
should_abort (*angr.analyses.forward_analysis.forward_analysis.AnalysisWorker property*), 764
should_add_successors (*angr.sim_procedure.SimProcedure property*), 1451
should_execute_statement() (*angr.annocfg.AnnotatedCFG method*), 957
should_skip_main_binary() (*angr.angrdb.serializers.loader.LoaderSerializer static method*), 954
should_take_exit() (*angr.annocfg.AnnotatedCFG method*), 957
should_use_hex() (*in module angr.utils.constants*), 1675
ShrBoundsConstraintHook (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 336
shrink() (*angr.analyses.reassembler.Data method*), 874
SideEffectStatement (*class in angr.ailment.statement*), 225
SideEffectStatementRewriter (*class in angr.rust.optimization_passes.utils*), 1430
SideEffectTag (*class in angr.knowledge_plugins.key_definitions.tag*), 1261
SIG_BLOCK (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
SIG_BLOCK (*angr.state_plugins.SimSystemPosix attribute*), 1536
SIG_SETMASK (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
SIG_SETMASK (*angr.state_plugins.SimSystemPosix attribute*), 1536
SIG_UNBLOCK (*angr.state_plugins.posix.SimSystemPosix attribute*), 1580
SIG_UNBLOCK (*angr.state_plugins.SimSystemPosix attribute*), 1536
sigaction (*class in angr.procedures.linux_kernel.sigaction*), 1367
sigaction (*class in angr.procedures.posix.sigaction*), 1375
SigBUS (*angr.protos.primitives_pb2.Edge attribute*), 1403
SigFPE (*angr.protos.primitives_pb2.Edge attribute*), 1403
SigFPE_IntDiv (*angr.protos.primitives_pb2.Edge attribute*), 1403
SigFPE_IntOvf (*angr.protos.primitives_pb2.Edge attribute*), 1404
SigILL (*angr.protos.primitives_pb2.Edge attribute*), 1404
sigmask() (*angr.state_plugins.posix.SimSystemPosix method*), 1581
sign_extend() (*in module angr.utils.bits*), 1674
SIGNATURES (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 1209
SIGNATURES (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
signed (*angr.procedures.stubs.format_parser.FormatSpecifier property*), 1379
signed (*angr.sim_type.SimTypeFloat attribute*), 1470
signed_int_type() (*in module angr.analyses.typehoon.typeconsts*), 913
SignedExtension (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.A attribute*), 334
sigprocmask (*class in angr.procedures.linux_kernel.sigprocmask*), 1367
sigprocmask() (*angr.state_plugins.posix.SimSystemPosix method*), 1582
SigSEGV (*angr.protos.primitives_pb2.Edge attribute*), 1404
SigTRAP (*angr.protos.primitives_pb2.Edge attribute*), 1404
sim_engine (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
sim_engine (*angr.state_plugins.InspectAttrs attribute*), 1499
sim_successors (*angr.state_plugins.inspect.InspectAttrs attribute*), 1568

sim_successors (*angr.state_plugins.InspectAttrs attribute*), 1499
SimAbstractMemoryError, 1083
SimAction (*class in angr.state_plugins.sim_action*), 1584
SimActionConstraint (*class in angr.state_plugins.sim_action*), 1585
SimActionData (*class in angr.state_plugins.sim_action*), 1585
SimActionError, 1085
SimActionExit (*class in angr.state_plugins.sim_action*), 1584
SimActionObject (*class in angr.state_plugins.sim_action_object*), 1585
SimActionOperation (*class in angr.state_plugins.sim_action*), 1585
SimActLocation (*class in angr.analyses.data_dep.sim_act_location*), 361
SimArrayArg (*class in angr.calling_conventions*), 966
SimCC (*class in angr.calling_conventions*), 967
SimCC.ArgSession (*class in angr.calling_conventions*), 968
SimCCAArch64 (*class in angr.calling_conventions*), 978
SimCCAArch64LinuxSyscall (*class in angr.calling_conventions*), 978
SimCCAArch64WindowsSyscall (*class in angr.calling_conventions*), 978
SimCCallError, 1085
SimCCAMD64LinuxSyscall (*class in angr.calling_conventions*), 976
SimCCAMD64WindowsSyscall (*class in angr.calling_conventions*), 976
SimCCARM (*class in angr.calling_conventions*), 976
SimCCARMHF (*class in angr.calling_conventions*), 977
SimCCARMLinuxSyscall (*class in angr.calling_conventions*), 977
SimCCARMWindowsSyscall (*class in angr.calling_conventions*), 977
SimCCCdecl (*class in angr.calling_conventions*), 971
SimCCError, 1086
SimCCGoAMD64 (*class in angr.calling_conventions*), 975
SimCCGoAMD64ABI0 (*class in angr.calling_conventions*), 976
SimCCM68k (*class in angr.engines.pcode.cc*), 1034
SimCCMicrosoftAMD64 (*class in angr.calling_conventions*), 973
SimCCMicrosoftCdecl (*class in angr.calling_conventions*), 972
SimCCMicrosoftFastcall (*class in angr.calling_conventions*), 972
SimCCMicrosoftThiscall (*class in angr.calling_conventions*), 972
SimCCN32 (*class in angr.calling_conventions*), 980
SimCCN32LinuxSyscall (*class in angr.calling_conventions*), 981
SimCCN64 (*class in angr.calling_conventions*), 980
SimCCN64LinuxSyscall (*class in angr.calling_conventions*), 981
SimCCO32 (*class in angr.calling_conventions*), 979
SimCCO32LinuxSyscall (*class in angr.calling_conventions*), 980
SimCCO64 (*in module angr.calling_conventions*), 980
SimCCPARISC (*class in angr.engines.pcode.cc*), 1035
SimCCPCodeBase (*class in angr.engines.pcode.cc*), 1034
SimCCPowerPC (*class in angr.calling_conventions*), 981
SimCCPowerPC (*class in angr.engines.pcode.cc*), 1035
SimCCPowerPC64 (*class in angr.calling_conventions*), 982
SimCCPowerPC64LinuxSyscall (*class in angr.calling_conventions*), 982
SimCCPowerPCLinuxSyscall (*class in angr.calling_conventions*), 981
SimCCRISCV (*class in angr.engines.pcode.cc*), 1035
SimCCRISCV64 (*class in angr.calling_conventions*), 979
SimCCRISCV64LinuxSyscall (*class in angr.calling_conventions*), 979
SimCCS390X (*class in angr.calling_conventions*), 982
SimCCS390XLinuxSyscall (*class in angr.calling_conventions*), 983
SimCCSH4 (*class in angr.engines.pcode.cc*), 1035
SimCCSoot (*class in angr.calling_conventions*), 982
SimCCSPARC (*class in angr.engines.pcode.cc*), 1035
SimCCStdcall (*class in angr.calling_conventions*), 972
SimCCSyscall (*class in angr.calling_conventions*), 973
SimCCSystemVAMD64 (*class in angr.calling_conventions*), 974
SimCCUnknown (*class in angr.calling_conventions*), 982
SimCCUsercall (*class in angr.calling_conventions*), 971
SimCCX86LinuxSyscall (*class in angr.calling_conventions*), 974
SimCCX86WindowsSyscall (*class in angr.calling_conventions*), 974
SimCCXtensa (*class in angr.engines.pcode.cc*), 1036
SimCGC (*class in angr.simos.cgc*), 1486
SimComboArg (*class in angr.calling_conventions*), 965
SimComboRegisterVariable (*class in angr.sim_variable*), 1479
SimConcreteBreakpointError, 1086
SimConcreteFilesystem (*class in angr.state_plugins.filesystem*), 1550
SimConcreteMemoryError, 1086
SimConcreteRegisterError, 1086
SimConcretizationStrategy (*class in angr.concretization_strategies.base*), 990
SimConcretizationStrategyAny (*class in angr.concretization_strategies.any*), 990
SimConcretizationStrategyAnyNamed (*class in*

*angr.concretization_strategies.any_named*), 990
SimConcretizationStrategyControlledData (*class in angr.concretization_strategies.controlled_data*), 991
SimConcretizationStrategyEval (*class in angr.concretization_strategies.eval*), 991
SimConcretizationStrategyLogging (*class in angr.concretization_strategies.logging*), 991
SimConcretizationStrategyMax (*class in angr.concretization_strategies.max*), 991
SimConcretizationStrategyNonzero (*class in angr.concretization_strategies.nonzero*), 992
SimConcretizationStrategyNonzeroRange (*class in angr.concretization_strategies.nonzero_range*), 992
SimConcretizationStrategyNorepeats (*class in angr.concretization_strategies.norepeats*), 992
SimConcretizationStrategyNorepeatsRange (*class in angr.concretization_strategies.norepeats_range*), 992
SimConcretizationStrategyRange (*class in angr.concretization_strategies.range*), 992
SimConcretizationStrategySignedAdd (*class in angr.concretization_strategies.signed_add*), 992
SimConcretizationStrategySingle (*class in angr.concretization_strategies.single*), 992
SimConcretizationStrategySolutions (*class in angr.concretization_strategies.solutions*), 992
SimConcretizationStrategyUnlimitedRange (*class in angr.concretization_strategies.unlimited_range*), 993
SimConstantVariable (*class in angr.sim_variable*), 1478
SimCppClass (*class in angr.sim_type*), 1475
SimCppClassValue (*class in angr.sim_type*), 1475
SimCppLibrary (*class in angr.procedures.definitions*), 1328
SimDebugVariable (*class in angr.state_plugins.debug_variables*), 1545
SimDebugVariablePlugin (*class in angr.state_plugins.debug_variables*), 1546
SimEmptyCallStackError, 1086
SimEngine (*class in angr.engines.engine*), 1004
SimEngineAILSimState (*class in angr.engines.ail.engine_light*), 1003
SimEngineDephiRewriting (*class in angr.analyses.decompiler.dephication.rewriting_engine*), 419
SimEngineError, 1085
SimEngineFactCollectorVEX (*class in angr.analyses.calling_convention.fact_collector*), 288
SimEngineFailure (*class in angr.engines.failure*), 1005
SimEngineFCPVEX (*class in angr.analyses.fcp.fcp*), 746
SimEngineInitFinderVEX (*class in angr.analyses.init_finder*), 786
SimEngineLight (*class in angr.engines.light.engine*), 1010
SimEngineLightAIL (*class in angr.engines.light.engine*), 1013
SimEngineLightVEX (*class in angr.engines.light.engine*), 1012
SimEngineNoexprAIL (*class in angr.engines.light.engine*), 1013
SimEngineNostmtAIL (*class in angr.engines.light.engine*), 1013
SimEngineNostmtVEX (*class in angr.engines.light.engine*), 1013
SimEnginePropagatorBaseMixin (*class in angr.analyses.propagator.engine_base*), 796
SimEnginePropagatorVEX (*class in angr.analyses.propagator.engine_vex*), 797
SimEngineRDAIL (*class in angr.analyses.reaching_definitions.engine_ail*), 838
SimEngineRDVEX (*class in angr.analyses.reaching_definitions.engine_vex*), 838
SimEngineSSARewriting (*class in angr.analyses.decompiler.ssailification.rewriting_engine*), 654
SimEngineSSATraversal (*class in angr.analyses.decompiler.ssailification.traversal_engine*), 656
SimEngineSyscall (*class in angr.engines.syscall*), 1065
SimEngineUnicorn (*class in angr.engines.unicorn*), 1065
SimEngineVRAIL (*class in angr.analyses.variable_recovery.engine_ail*), 923
SimEngineVRBase (*class in angr.analyses.variable_recovery.engine_base*), 924
SimEngineVRVEX (*class in angr.analyses.variable_recovery.engine_vex*), 925
SimEngineXRefsVEX (*class in angr.analyses.xrefs*), 937
SimError, 1083
SimEvent (*class in angr.state_plugins.sim_event*), 1587
SimEventError, 1084
SimException, 1086
SimExpressionError, 1084
SimFastMemoryError, 1084
SimFastPathError, 1085
SimFile (*class in angr.storage.file*), 1611
SimFileBase (*class in angr.storage.file*), 1610

SimFileDescriptor (*class in angr.storage.file*), 1615
SimFileDescriptorBase (*class in angr.storage.file*), 1613
SimFileDescriptorDuplex (*class in angr.storage.file*), 1616
SimFileError, 1084
SimFileStream (*class in angr.storage.file*), 1612
SimFilesystem (*class in angr.state_plugins.filesystem*), 1548
SimFilesystemError, 1084
SimFunctionArgument (*class in angr.calling_conventions*), 963
simgr() (*angr.factory.AngrObjectFactory method*), 1112
SimHeapBase (*class in angr.state_plugins.heap.heap_base*), 1555
SimHeapBrk (*class in angr.state_plugins.heap.heap_brk*), 1556
SimHeapError, 1084
SimHeapFreelist (*class in angr.state_plugins.heap.heap_freelist*), 1558
SimHeapLibc (*class in angr.state_plugins.heap.heap_libc*), 1558
SimHeapPTMalloc (*class in angr.state_plugins.heap.heap_ptmalloc*), 1560
SimHostFilesystem (*class in angr.state_plugins.filesystem*), 1550
similar_conditional_when_single_corrected() (*angr.analyses.decompiler.optimization_passes.duplication_reverter.DuplicationReverter method*), 571
similarity() (*angr.exploration_techniques.unique.UniqueSearch static method*), 1109
SimInspectMixin (*class in angr.engines.vex.heavy.inspect*), 1078
SimInspector (*class in angr.state_plugins.inspect*), 1571
SimIROp (*class in angr.engines.vex.claripy.irop*), 1073
SimIRSBError, 1085
SimIRSBNoDecodeError, 1085
SimJavaVM (*class in angr.simos.javavm*), 1486
SimJavaVmClassloader (*class in angr.state_plugins.javavm_classloader*), 1573
SimLabeledMemoryObject (*class in angr.storage.memory_object*), 1667
SimLibrary (*class in angr.procedures.definitions*), 1325
SimLightRegisters (*class in angr.state_plugins.light_registers*), 1577
SimLightState (*class in angr.slicer*), 1493
SimLinux (*class in angr.simos.linux*), 1488
SimLyingRegArg (*class in angr.calling_conventions*), 971
SimMemoryAddressError, 1084
SimMemoryError, 1083
SimMemoryLimitError, 1084
SimMemoryMissingError, 1083
SimMemoryObject (*class in angr.storage.memory_object*), 1666
SimMemoryVariable (*class in angr.sim_variable*), 1479
SimMemView (*class in angr.state_plugins.view*), 1604
SimMergeError, 1083
SimMissingTempError, 1085
SimMount (*class in angr.state_plugins.filesystem*), 1549
SimOperationError, 1084
SimOS (*class in angr.simos.simos*), 1488
SIMOS_CGC (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 1602
SIMOS_LINUX (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 1602
SIMOS_OTHER (*angr.state_plugins.unicorn_engine.SimOSEnum attribute*), 1602
SimOSEnum (*class in angr.state_plugins.unicorn_engine*), 1602
SimPackets (*class in angr.storage.file*), 1612
SimPacketsSlots (*class in angr.storage.file*), 1616
SimPacketsStream (*class in angr.storage.file*), 1613
SimpleInterfaceMixin (*class in angr.storage.memory_mixins.simple_interface_mixin*), 1664
SimpleSolver (*class in angr.analyses.typehoon.simple_solver*), 902
SimplificationMixin (*class in angr.storage.memory_mixins.simplification_mixin*), 1665
simplified_data_graph (*angr.analyses.ddg.DDG property*), 365
simplified_graph (*angr.analyses.data_dep.data_dependency_analysis.D property*), 358
SimplifierAILEngine (*class in angr.analyses.decompiler.optimization_passes.engine_base*), 575
SimplifierAILState (*class in angr.analyses.decompiler.optimization_passes.engine_base*), 574
simplify() (*angr.sim_state.SimState method*), 1454
simplify() (*angr.state_hierarchy.StateHierarchy method*), 1494
simplify() (*angr.state_plugins.solver.SimSolver method*), 1595
simplify_condition() (*angr.analyses.decompiler.condition_processor.ConditionProcess static method*), 410
simplify_lowered_switches() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl* 641
simplify_lowered_switches_core() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl* 641

simplify_switch_clusters() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_optimizer*), 640
SimplifyBitwiseInserts (*class in angr.analyses.decompiler.peephole_optimizations.simplify_bitwise_ops*), 605
SimplifyPcRelativeLoads (*class in angr.analyses.decompiler.peephole_optimizations.simplify_pc_relative_loads*), 620
SimPosixError, 1084
SIMPROC (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 1209
SIMPROC (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
SIMPROCEDURE (*angr.state_plugins.inspect.EventType attribute*), 1565
simprocedure (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
simprocedure (*angr.state_plugins.InspectAttrs attribute*), 1499
SimProcedure (*class in angr.sim_procedure*), 1448
simprocedure_addr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
simprocedure_addr (*angr.state_plugins.InspectAttrs attribute*), 1499
simprocedure_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
simprocedure_name (*angr.state_plugins.InspectAttrs attribute*), 1499
simprocedure_result (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
simprocedure_result (*angr.state_plugins.InspectAttrs attribute*), 1499
SimProcedureArgumentError, 1085
SimProcedureError, 1085
SimReferenceArgument (*class in angr.calling_conventions*), 966
SimRegArg (*class in angr.calling_conventions*), 964
SimRegionMapError, 1084
SimRegisterVariable (*class in angr.sim_variable*), 1478
SimRegNameView (*class in angr.state_plugins.view*), 1604
SimReliftException, 1085
SimSegfaultError (*in module angr.errors*), 1086
SimSegfaultException, 1086
SimShadowStackError, 1085
SimSlicer (*class in angr.slicer*), 1493
SimSlicerError, 1085
SimSnimmucNxp (*class in angr.simos.snimmuc_nxp*), 1491
SimSolver (*class in angr.state_plugins.solver*), 1587
SimSolverError, 1084
SimSolverModeError, 1084
SimSolverOptionError, 1084
SimSootExpr (*class in angr.engines.soot.expressions.base*), 1050
SimSootExpr_ArrayRef (*class in angr.engines.soot.expressions.arrayref*), 1050
SimSootExpr_Binop (*class in angr.engines.soot.expressions.binop*), 1050
SimSootExpr_Cast (*class in angr.engines.soot.expressions.cast*), 1050
SimSootExpr_ClassConstant (*class in angr.engines.soot.expressions.constants*), 1051
SimSootExpr_Condition (*class in angr.engines.soot.expressions.condition*), 1050
SimSootExpr_DoubleConstant (*class in angr.engines.soot.expressions.constants*), 1050
SimSootExpr_FloatConstant (*class in angr.engines.soot.expressions.constants*), 1050
SimSootExpr_InstanceFieldRef (*class in angr.engines.soot.expressions.instancefieldref*), 1051
SimSootExpr_InstanceOf (*class in angr.engines.soot.expressions.instanceOf*), 1051
SimSootExpr_IntConstant (*class in angr.engines.soot.expressions.constants*), 1050
SimSootExpr_InterfaceInvoke (*class in angr.engines.soot.expressions.invoke*), 1051
SimSootExpr_Length (*class in angr.engines.soot.expressions.length*), 1051
SimSootExpr_Local (*class in angr.engines.soot.expressions.local*), 1051
SimSootExpr_LongConstant (*class in angr.engines.soot.expressions.constants*), 1050
SimSootExpr_New (*class in angr.engines.soot.expressions.new*), 1052
SimSootExpr_NewArray (*class in angr.engines.soot.expressions.newArray*), 1052
SimSootExpr_NewMultiArray (*class in angr.engines.soot.expressions.newMultiArray*), 1052
SimSootExpr_NullConstant (*class in angr.engines.soot.expressions.constants*), 1051
SimSootExpr_ParamRef (*class in angr.engines.soot.expressions.paramref*),

1052
SimSootExpr_Phi (*class in angr.engines.soot.expressions.phi*), 1052
SimSootExpr_SpecialInvoke (*class in angr.engines.soot.expressions.invoke*), 1051
SimSootExpr_StaticFieldRef (*class in angr.engines.soot.expressions.staticfieldref*), 1052
SimSootExpr_StaticInvoke (*class in angr.engines.soot.expressions.invoke*), 1051
SimSootExpr_StringConstant (*class in angr.engines.soot.expressions.constants*), 1050
SimSootExpr_ThisRef (*class in angr.engines.soot.expressions.thisref*), 1052
SimSootExpr_Unsupported (*class in angr.engines.soot.expressions.unsupported*), 1052
SimSootExpr_VirtualInvoke (*class in angr.engines.soot.expressions.invoke*), 1051
SimSootStmt (*class in angr.engines.soot.statements.base*), 1054
SimSootStmt_Assign (*class in angr.engines.soot.statements.assign*), 1054
SimSootStmt_Goto (*class in angr.engines.soot.statements.goto*), 1054
SimSootStmt_Identity (*class in angr.engines.soot.statements.identity*), 1054
SimSootStmt_If (*class in angr.engines.soot.statements.if_*), 1054
SimSootStmt_Invoke (*class in angr.engines.soot.statements.invoke*), 1054
SimSootStmt_LookupSwitch (*class in angr.engines.soot.statements.switch*), 1055
SimSootStmt_Return (*class in angr.engines.soot.statements.return_*), 1054
SimSootStmt_ReturnVoid (*class in angr.engines.soot.statements.return_*), 1055
SimSootStmt_TableSwitch (*class in angr.engines.soot.statements.switch*), 1055
SimSootStmt_Throw (*class in angr.engines.soot.statements.throw*), 1055
SimSootValue (*class in angr.engines.soot.values.base*), 1060
SimSootValue_ArrayBaseRef (*class in angr.engines.soot.values.arrayref*), 1059
SimSootValue_ArrayRef (*class in angr.engines.soot.values.arrayref*), 1059
SimSootValue_InstanceFieldRef (*class in angr.engines.soot.values.instancefieldref*), 1060
SimSootValue_IntConstant (*class in angr.engines.soot.values.constants*), 1060
SimSootValue_Local (*class in angr.engines.soot.values.local*), 1060
SimSootValue_ParamRef (*class in angr.engines.soot.values.paramref*), 1061
SimSootValue_StaticFieldRef (*class in angr.engines.soot.values.staticfieldref*), 1061
SimSootValue_StringRef (*class in angr.engines.soot.values.strref*), 1061
SimSootValue_ThisRef (*class in angr.engines.soot.values.thisref*), 1062
SimStackArg (*class in angr.calling_conventions*), 964
SimStackVariable (*class in angr.sim_variable*), 1479
SimState (*class in angr.sim_state*), 1452
SimStateCGC (*class in angr.state_plugins.cgc*), 1544
SimStateEdgeHitmap (*class in angr.state_plugins.edge_hitmap*), 1547
SimStateError, 1083
SimStateGlobals (*class in angr.state_plugins.globals*), 1551
SimStateHistory (*class in angr.state_plugins.history*), 1561
SimStateIcicle (*class in angr.state_plugins.icicle*), 1564
SimStateJNIReferences (*class in angr.state_plugins.jni_references*), 1573
SimStateLibc (*class in angr.state_plugins.libc*), 1574
SimStateLog (*class in angr.state_plugins.log*), 1577
SimStateLoopData (*class in angr.state_plugins.loop_data*), 1577
SimStateMarker (*class in angr.analyses.pathfinder*), 794
SimStatementError, 1085
SimStateOptions (*class in angr.sim_state_options*), 1456
SimStateOptionsError, 1086
SimStatePlugin (*class in angr.state_plugins.plugin*), 1578
SimStatePreconstrainer (*class in angr.state_plugins.preconstrainer*), 1582
SimStateScratch (*class in angr.state_plugins.scratch*), 1583
SimStateStorageMixin (*class in angr.engines.vex.heavy.heavy*), 1078
SimStruct (*class in angr.sim_type*), 1471
SimStructArg (*class in angr.calling_conventions*), 965
SimStructValue (*class in angr.sim_type*), 1471
SimSuccessors (*class in angr.engines.successors*), 1063
SimSymbolicFilesystemError, 1084
SimSymbolizer (*class in angr.state_plugins.symbolizer*), 1596
SimSyscallLibrary (*class in angr.procedures.definitions*), 1330
SimSystemPosix (*class in angr.state_plugins.posix*), 1579
SimTemporaryVariable (*class in angr.sim_variable*),

1478
SimTranslationError, 1085
SimType (*class in angr.sim_type*), 1459
simtype2tc() (*angr.analyses.typehoon.translator.TypeTranslator method*), 906
SimTypeArray (*class in angr.sim_type*), 1467
SimTypeBitfield (*class in angr.sim_type*), 1473
SimTypeBool (*class in angr.sim_type*), 1465
SimTypeBottom (*class in angr.sim_type*), 1462
SimTypeChar (*class in angr.sim_type*), 1464
SimTypeCollection (*class in angr.procedures.definitions*), 1324
SimTypeCppFunction (*class in angr.sim_type*), 1469
SimTypeDouble (*class in angr.sim_type*), 1470
SimTypeEnum (*class in angr.sim_type*), 1472
SimTypeFd (*class in angr.sim_type*), 1465
SimTypeFixedSizeArray (*in module angr.sim_type*), 1468
SimTypeFixedSizeInt (*class in angr.sim_type*), 1464
SimTypeFloat (*class in angr.sim_type*), 1470
SimTypeFunction (*class in angr.sim_type*), 1469
SimTypeInt (*class in angr.sim_type*), 1463
SimTypeInt128 (*class in angr.sim_type*), 1464
SimTypeInt256 (*class in angr.sim_type*), 1464
SimTypeInt512 (*class in angr.sim_type*), 1464
SimTypeLength (*class in angr.sim_type*), 1470
SimTypeLong (*class in angr.sim_type*), 1464
SimTypeLongLong (*class in angr.sim_type*), 1464
SimTypeNum (*class in angr.sim_type*), 1462
SimTypeNumOffset (*class in angr.sim_type*), 1475
SimTypePointer (*class in angr.sim_type*), 1466
SimTypeRef (*class in angr.sim_type*), 1475
SimTypeReference (*class in angr.sim_type*), 1467
SimTypeReg (*class in angr.sim_type*), 1462
SimTypeShort (*class in angr.sim_type*), 1464
SimTypeString (*class in angr.sim_type*), 1468
SimTypeTempRef (*class in angr.analyses.typehoon.translator*), 905
SimTypeTop (*class in angr.sim_type*), 1462
SimTypeWideChar (*class in angr.sim_type*), 1465
SimTypeWString (*class in angr.sim_type*), 1468
SimUCManager (*class in angr.state_plugins.uc_manager*), 1598
SimUCManagerAllocationError, 1086
SimUCManagerError, 1086
SimUefi (*class in angr.simos.uefi*), 1491
simulation_manager() (*angr.factory.AngrObjectFactory method*), 1112
SimulationManager (*class in angr.sim_manager*), 1441
SimulationManagerError, 1081
SimUnicornError, 1086
SimUnicornSymbolic, 1086
SimUnicornUnsupport, 1086
SimUninitializedAccessError, 1085
SimUnion (*class in angr.sim_type*), 1472
SimUnionValue (*class in angr.sim_type*), 1472
SimUnsatError, 1084
SimUnsupportedError, 1084
SimUserland (*class in angr.simos.userland*), 1491
SimValueError, 1084
simvar_from_bytes_polymorphic() (*in module angr.utils.ail_serialization*), 1671
simvar_pool (*angr.analyses.decompiler.structured_codegen.c_serialize.Se property*), 689
simvar_to_bytes_polymorphic() (*in module angr.utils.ail_serialization*), 1671
SimVariable (*class in angr.sim_variable*), 1477
SimWindows (*class in angr.simos.windows*), 1492
SimXbox (*class in angr.simos.xbox*), 1493
SimZeroDivisionException, 1086
single_valued() (*angr.state_plugins.solver.SimSolver method*), 1595
SingleBitCondToBoolExpr (*class in angr.analyses.decompiler.peephole_optimizations.single_bit_cond*), 620
SingleBitXor (*class in angr.analyses.decompiler.peephole_optimizations.single_bit_xor*), 621
SingleExpressionCounter (*class in angr.analyses.decompiler.counters.expression_counters*), 412
SingleNodeGraphVisitor (*class in angr.analyses.forward_analysis.visitors.single_node_graph*), 769
SInt16 (*class in angr.analyses.typehoon.typeconsts*), 908
SInt32 (*class in angr.analyses.typehoon.typeconsts*), 908
SInt64 (*class in angr.analyses.typehoon.typeconsts*), 908
SInt8 (*class in angr.analyses.typehoon.typeconsts*), 908
size (*angr.analyses.decompiler.known_patterns.dsl.PLoad attribute*), 496
size (*angr.analyses.decompiler.known_patterns.dsl.PStackField attribute*), 500
size (*angr.analyses.decompiler.known_patterns.dsl.PStore attribute*), 503
size (*angr.analyses.decompiler.known_patterns.PLoad attribute*), 473
size (*angr.analyses.decompiler.known_patterns.PStackField attribute*), 475
size (*angr.analyses.decompiler.known_patterns.PStore attribute*), 476
size (*angr.analyses.typehoon.simple_solver.SketchNode property*), 900
size (*angr.analyses.typehoon.typeconsts.Array property*), 910
SIZE (*angr.analyses.typehoon.typeconsts.BottomType attribute*), 907
size (*angr.analyses.typehoon.typeconsts.Enum prop-*

*erty*), 912
SIZE (*angr.analyses.typehoon.typeconsts.Fd attribute*), 908
SIZE (*angr.analyses.typehoon.typeconsts.Float32 attribute*), 909
SIZE (*angr.analyses.typehoon.typeconsts.Float64 attribute*), 909
SIZE (*angr.analyses.typehoon.typeconsts.Int1 attribute*), 907
SIZE (*angr.analyses.typehoon.typeconsts.Int128 attribute*), 909
SIZE (*angr.analyses.typehoon.typeconsts.Int16 attribute*), 907
SIZE (*angr.analyses.typehoon.typeconsts.Int256 attribute*), 909
SIZE (*angr.analyses.typehoon.typeconsts.Int32 attribute*), 908
SIZE (*angr.analyses.typehoon.typeconsts.Int512 attribute*), 909
SIZE (*angr.analyses.typehoon.typeconsts.Int64 attribute*), 908
SIZE (*angr.analyses.typehoon.typeconsts.Int8 attribute*), 907
size (*angr.analyses.typehoon.typeconsts.IntVar property*), 909
size (*angr.analyses.typehoon.typeconsts.RustEnum property*), 911
size (*angr.analyses.typehoon.typeconsts.Struct property*), 911
SIZE (*angr.analyses.typehoon.typeconsts.TopType attribute*), 907
SIZE (*angr.analyses.typehoon.typeconsts.TypeConstant attribute*), 906
size (*angr.analyses.typehoon.typeconsts.TypeConstant property*), 907
size (*angr.block.CapstoneInsn property*), 960
size (*angr.block.DisassemblerInsn property*), 959
size (*angr.block.PCodeInsn property*), 960
size (*angr.block.SootBlock property*), 961
size (*angr.engines.pcode.lifter.IRSB property*), 1040
size (*angr.engines.pcode.lifter.PcodeDisassemblerInsn property*), 1037
size (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
size (*angr.knowledge_plugins.functions.function.Function property*), 1213
size (*angr.knowledge_plugins.key_definitions.definition.Definition property*), 1247
size (*angr.procedures.stubs.format_parser.FormatSpecifier property*), 1379
size (*angr.rust.sim_type.EnumVariant property*), 1434
size (*angr.rust.sim_type.RustSimEnum property*), 1434
size (*angr.rust.sim_type.RustSimStruct property*), 1433
size (*angr.rust.sim_type.RustSimTypeArray property*), 1433
size (*angr.rust.sim_type.RustSimTypeBottom property*), 1434
size (*angr.rust.sim_type.RustSimTypeFunction property*), 1432
size (*angr.rust.sim_type.RustSimTypeInt property*), 1431
size (*angr.rust.sim_type.RustSimTypeReference property*), 1432
size (*angr.rust.sim_type.RustSimTypeSize property*), 1431
size (*angr.rust.sim_type.RustSimTypeSlice property*), 1433
size (*angr.rust.sim_type.RustSimTypeUnit property*), 1435
size (*angr.rustylib.Segment attribute*), 1439
size (*angr.sim_type.SimCppClass property*), 1475
size (*angr.sim_type.SimStruct property*), 1471
size (*angr.sim_type.SimType property*), 1460
size (*angr.sim_type.SimTypeArray property*), 1467
size (*angr.sim_type.SimTypeBitfield property*), 1474
size (*angr.sim_type.SimTypeDouble property*), 1471
size (*angr.sim_type.SimTypeEnum property*), 1473
size (*angr.sim_type.SimTypeFd property*), 1466
size (*angr.sim_type.SimTypeFixedSizeInt property*), 1464
size (*angr.sim_type.SimTypeFloat property*), 1470
size (*angr.sim_type.SimTypeFunction property*), 1469
size (*angr.sim_type.SimTypeInt property*), 1463
size (*angr.sim_type.SimTypeLength property*), 1470
size (*angr.sim_type.SimTypeNum property*), 1463
size (*angr.sim_type.SimTypePointer property*), 1466
size (*angr.sim_type.SimTypeReference property*), 1467
size (*angr.sim_type.SimTypeString property*), 1468
size (*angr.sim_type.SimTypeWString property*), 1469
size (*angr.sim_type.SimUnion property*), 1472
size (*angr.sim_type.TypeRef property*), 1461
size (*angr.storage.file.SimFile property*), 1612
size (*angr.storage.file.SimFileBase property*), 1611
size (*angr.storage.file.SimPackets property*), 1613
size (*angr.storage.file.SimPacketsSlots property*), 1617
size() (*angr.calling_conventions.AllocHelper method*), 963
size() (*angr.storage.file.SimFileDescriptorBase method*), 1615
size() (*angr.storage.memory_object.SimMemoryObject method*), 1667
size_t_typename() (*in module angr.analyses.decompiler.known_patterns.context*), 487
SizeConcretizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 1665
SizeNaming (*class in angr.analyses.decompiler.semantic_naming.size_nam*
652

SizeNormalizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 1665
Sketch (*class in angr.analyses.typehoon.simple_solver*), 900
SketchNode (*class in angr.analyses.typehoon.simple_solver*), 900
SketchNodeBase (*class in angr.analyses.typehoon.simple_solver*), 899
skip_conversions (*angr.analyses.decompiler.known_patterns.ExpressionMassage attribute*), 491
SkipConversionNotice, 216
skipped_converts (*angr.analyses.decompiler.known_patterns.FunctionMassage attribute*), 490
sleep (*class in angr.procedures.posix.sleep*), 1375
slice (*angr.blade.Blade property*), 959
slice_callgraph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 350
slice_cfg_graph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 350
slice_function_graph() (*in module angr.analyses.cfg_slice_to_sink.graph*), 350
Slicecutor (*class in angr.exploration_techniques.slicecutor*), 1104
SLivenessAnalysis (*class in angr.analyses.s_liveness*), 878
SLivenessModel (*class in angr.analyses.s_liveness*), 878
SlottedMemoryMixin (*class in angr.storage.memory_mixins.slotted_memory*), 1665
SmartFindMixin (*class in angr.storage.memory_mixins.smart_find_mixin*), 1666
SmartLRUCache (*class in angr.utils.smart_cache*), 1695
SMod() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1586
snapshot_successors() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 630
snapshot_user_edits() (*in module angr.analyses.decompiler.edits.cache*), 436
snippet() (*angr.factory.AngrObjectFactory method*), 1110
snprintf (*class in angr.analyses.identifier.functions.snprintf*), 779
snprintf (*class in angr.procedures.libc.snprintf*), 1359
socket (*class in angr.procedures.posix.socket*), 1375
solution() (*angr.state_plugins.solver.SimSolver method*), 1590
solve() (*angr.analyses.typehoon.simple_solver.SimpleSolver method*), 903
soot (*angr.block.SootBlock property*), 961
SootBlock (*class in angr.block*), 961
SootBlockNode (*class in angr.codenode*), 987
SootClassHierarchy (*class in angr.analyses.soot_class_hierarchy*), 893
SootClassHierarchyError, 893
SootExpression (*class in angr.analyses.disassembly*), 741
SootExpressionInvoke (*class in angr.analyses.disassembly*), 741
SootExpressionStaticFieldRef (*class in angr.analyses.disassembly*), 741
SootExpressionTarget (*class in angr.analyses.disassembly*), 741
SootFieldNotLoadedException, 1047
SootFunctionState (*class in angr.knowledge_plugins.functions.soot_function*), 1228
SootMethodNotLoadedException, 1047
SootMixin (*class in angr.engines.soot.engine*), 1047
SootStatement (*class in angr.analyses.disassembly*), 741
sort (*angr.rustylib.Segment attribute*), 1439
sort() (*angr.exploration_techniques.spiller.PickledStatesBase method*), 1104
sort_edge() (*angr.utils.graph.GraphUtils static method*), 1687
sort_key (*angr.ailment.Block attribute*), 210
sort_key (*angr.ailment.block.Block attribute*), 215
sort_local_vars() (*angr.analyses.decompiler.structured_codegen.c.CFu static method*), 673
sort_node() (*angr.utils.graph.GraphUtils static method*), 1687
sort_nodes() (*angr.analyses.forward_analysis.visitors.graph.GraphVisito method*), 767
sort_nodes_by_order() (*angr.utils.graph.DirectedGraphHelper method*), 1689
sort_ratio_backwards() (*angr.rustylib.SegmentList method*), 1440
sorted_bookmarks() (*angr.knowledge_plugins.bookmarks.Bookmarks method*), 1150
SortedDict (*class in angr.knowledge_plugins.variables.variable_manager* 1283
sources_defns (*angr.analyses.reaching_definitions.function_handler.Func attribute*), 839
sp_offset() (*angr.engines.light.engine.SimEngineLight static method*), 1011
spec_type (*angr.procedures.stubs.format_parser.FormatSpecifier property*), 1379
Special (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 741
SPECIAL_THUNKS (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 321
SPECIAL_THUNKS (*angr.analyses.cfg.CFGFast attribute*), 299
SPECIAL_THUNKS (*angr.analyses.CFGFast attribute*),

242
SpecialFillerMixin (class in angr.storage.memory_mixins.default_filler_mixin), 1635
spill_down_edges() (angr.knowledge_plugins.cfg.spilling_digraph.SpillingDiGraph method), 1189
spilled_count (angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG property), 1186
spilled_count (angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeDict property), 1180
spilled_count (angr.knowledge_plugins.functions.function_manager.SpillingFunctionDict property), 1220
spilled_function_count (angr.knowledge_plugins.functions.function_manager.FunctionManager property), 1227
Spiller (class in angr.exploration_techniques.spiller), 1105
SpillingAdjDict (class in angr.knowledge_plugins.cfg.spilling_digraph), 1187
SpillingCFG (class in angr.knowledge_plugins.cfg.spilling_cfg), 1181
SpillingCFGNodeDict (class in angr.knowledge_plugins.cfg.spilling_cfg), 1179
SpillingDecompilationDict (class in angr.knowledge_plugins.structured_code), 1278
SpillingDiGraph (class in angr.knowledge_plugins.cfg.spilling_digraph), 1188
SpillingFunctionDict (class in angr.knowledge_plugins.functions.function_manager), 1219
SpillingVariableInternalDict (class in angr.knowledge_plugins.variables.spilling_vardict), 1282
split() (angr.keyed_region.RegionObject method), 1116
split() (angr.sim_manager.SimulationManager method), 1446
split_ail_block() (angr.analyses.decompiler.optimization_passes.duplication_reverter.ail_merge_graph.AILBlockSplit static method), 568
split_ail_block() (in module angr.analyses.decompiler.known_patterns.block_split), 486
split_arm_op_string() (angr.analyses.disassembly.Instruction static method), 741
split_block() (angr.rust.mixins.cfg_transformation_mixin.CFGTransformationMixin method), 1418
split_op_string() (angr.analyses.disassembly.Instruction static method), 741
split_operands() (in module angr.analyses.reassembler), 870
SpOffset (class in angr.engines.light.data), 1010
sprintf (class in angr.analyses.identifier.functions.sprintf), 780
sprintf (class in angr.procedures.libc.sprintf), 1359
sprop_cache_scope() (in module angr.analyses.s_propagator), 879
SPropagator (class in angr.analyses.s_propagator), 879
SPropagatorAnalysis (class in angr.analyses.s_propagator), 880
SPropagatorModel (class in angr.analyses.s_propagator), 879
squash_array_reference() (in module angr.analyses.decompiler.structured_codegen.rust), 691
squash_array_reference() (in module angr.utils.types), 1701
srand (class in angr.procedures.libc.srand), 1359
SRDAMixin (class in angr.rust.mixins.srda_mixin), 1419
SRDAModel (class in angr.analyses.s_reaching_definitions.s_rda_model), 885
SRDAView (class in angr.analyses.s_reaching_definitions.s_rda_view), 889
SReachingDefinitions (class in angr.analyses.s_reaching_definitions.s_reaching_definitions), 891
SReachingDefinitionsAnalysis (class in angr.analyses.s_reaching_definitions.s_reaching_definitions), 891
SSA_LEVEL0_TRANSFORMATION (angr.analyses.decompiler.clinic.ClinicStage attribute), 404
SSA_LEVEL1_TRANSFORMATION (angr.analyses.decompiler.clinic.ClinicStage attribute), 404
Ssailification (class in angr.analyses.decompiler.ssailification.ssailification), 655
SSAVariableMixin (class in angr.rust.mixins.ssa_variable_mixin), 1420
sscanf (class in angr.procedures.libc.sscanf), 1360
sse_extend() (angr.calling_conventions.SimRegArg method), 964
stable (angr.analyses.decompiler.known_patterns.wdk_shared_data.Ksud attribute), 534
stable_hash() (in module angr.utils.hashing), 1689
StableVarExprHasher (class in angr.analyses.decompiler.optimization_passes.lowered_switch_si 584
stack (angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsSta property), 862
stack (angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemo property), 1636

stack_actions (*angr.state_plugins.history.SimStateHistory property*), 1562
stack_addr_from_offset() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 929
stack_address() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 861
stack_address() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase method*), 928
stack_address() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1251
STACK_ALIGNMENT (*angr.calling_conventions.SimCC attribute*), 967
STACK_ALIGNMENT (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
STACK_ALIGNMENT (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
STACK_ALIGNMENT (*angr.calling_conventions.SimCCRISCV64 attribute*), 979
STACK_ALIGNMENT (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 975
STACK_ALIGNMENT (*angr.SimCC attribute*), 173
stack_base (*angr.storage.memory_mixins.regioned_memory.region_data.RegionMap property*), 1660
STACK_CANARY (*angr.analyses.decompiler.stack_item.StackItemType attribute*), 659
stack_id() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 1664
stack_loc() (*angr.calling_conventions.AllocHelper class method*), 963
stack_offset (*angr.analyses.decompiler.edits.resolve.ResolvedVariable attribute*), 447
stack_offset (*angr.analyses.decompiler.edits.ResolvedVariable attribute*), 428
stack_offset (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation property*), 1244
stack_offset (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 1245
stack_offset_to_stack_addr() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions method*), 1252
stack_pointer_as_atom() (*angr.analyses.reaching_definitions.function_handler_library.FunctionHandlerLibrary static method*), 845
stack_pop() (*angr.sim_state.SimState method*), 1455
stack_push() (*angr.sim_state.SimState method*), 1455
stack_read() (*angr.analyses.calling_convention.fact_collector.FactCollectorState method*), 287
stack_read() (*angr.analyses.fcp.fcp.FCPState method*), 745
stack_read() (*angr.sim_state.SimState method*), 1455
stack_slot_fn (*angr.analyses.decompiler.known_patterns.dsl.Match attribute*), 491
stack_space() (*angr.calling_conventions.SimCC method*), 969
stack_suffix() (*angr.state_plugins.callstack.CallStack method*), 1544
stack_suffix() (*angr.state_plugins.callstack.CallStack static method*), 1543
stack_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
stack_written() (*angr.analyses.calling_convention.fact_collector.FactCollectorState method*), 288
stack_written() (*angr.analyses.fcp.fcp.FCPState method*), 745
StackAllocationMixin (*class in angr.storage.memory_mixins.paged_memory.stack_allocation_mixin*), 1655
STACKARG_SP_BUFF (*angr.calling_conventions.SimCC attribute*), 967
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCN64 attribute*), 980
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCO32 attribute*), 977
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCPowerPC attribute*), 981
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCPowerPC64 attribute*), 982
STACKARG_SP_BUFF (*angr.calling_conventions.SimCCS390X attribute*), 983
STACKARG_SP_BUFF (*angr.engines.pcode.cc.SimCCPowerPC attribute*), 1036
STACKARG_SP_BUFF (*angr.SimCC attribute*), 173
STACKARG_SP_DIFF (*angr.calling_conventions.SimCC attribute*), 967
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCCdecl attribute*), 971
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 972
STACKARG_SP_DIFF (*angr.calling_conventions.SimCCSystemVAMD64 attribute*), 974
STACKARG_SP_DIFF (*angr.engines.pcode.cc.SimCCM68k attribute*), 1034
STACKARG_SP_DIFF (*angr.SimCC attribute*), 173
StackArgOffsetManager (*class in angr.analyses.decompiler.stackarg_offset_manager*), 660
StackBaseOffset (*class in angr.ailment.expression*), 222
StackCanarySimplifier (*class in angr.analyses.decompiler.optimization_passes.stack_canary_simp*

594
StackDefinition (*class in angr.rust.mixins.dfa_mixin*), 1418
StackItem (*class in angr.analyses.decompiler.stack_item*), 659
StackItem (*class in angr.protos.clinic_pb2*), 1397
StackItemType (*class in angr.analyses.decompiler.stack_item*), 659
StackLocationAnnotation (*class in angr.analyses.variable_recovery.annotations*), 923
StackPointerTracker (*class in angr.analyses.stack_pointer_tracker*), 895
StackPointerTrackerState (*class in angr.analyses.stack_pointer_tracker*), 894
stackvar_get() (*angr.analyses.decompiler.ssailification.traversal_engine.SimEngineSSATraversal method*), 657
stackvar_set() (*angr.analyses.decompiler.ssailification.traversal_engine.SimEngineSSATraversal method*), 657
stackvar_unify() (*angr.analyses.decompiler.ssailification.traversal_state.TraversalState method*), 658
StackVariable (*class in angr.protos.variables_pb2*), 1405
StackvarMaxSizeEntry (*class in angr.protos.decompilation_cache_pb2*), 1402
StackVVarPredicate (*class in angr.analyses.s_reaching_definitions.s_rda_view*), 889
STAGE (*angr.analyses.decompiler.optimization_passes.base_ptr_save_simplifier.BasePtrSaveSimplifier attribute*), 560
STAGE (*angr.analyses.decompiler.optimization_passes.BaseOptimizationPass attribute*), 540
STAGE (*angr.analyses.decompiler.optimization_passes.call_stmt_rewriter.CallStatementRewriter attribute*), 560
STAGE (*angr.analyses.decompiler.optimization_passes.CallStatementRewriter attribute*), 540
STAGE (*angr.analyses.decompiler.optimization_passes.code_motion.CodeMotionOptimization attribute*), 561
STAGE (*angr.analyses.decompiler.optimization_passes.CodeMotionOptimization attribute*), 541
STAGE (*angr.analyses.decompiler.optimization_passes.condition_constprop.ConditionConstantPropagation attribute*), 562
STAGE (*angr.analyses.decompiler.optimization_passes.ConditionConstantPropagation attribute*), 542
STAGE (*angr.analyses.decompiler.optimization_passes.const_derefs.ConstantDereferencesSimplifier attribute*), 563
STAGE (*angr.analyses.decompiler.optimization_passes.const_prop_reverter.ConstPropOptReverter attribute*), 563
STAGE (*angr.analyses.decompiler.optimization_passes.ConstantDereferencesSimplifier attribute*), 543
STAGE (*angr.analyses.decompiler.optimization_passes.ConstPropOptReverter attribute*), 542
STAGE (*angr.analyses.decompiler.optimization_passes.cross_jump_reverter.CrossJumpReverter attribute*), 564
STAGE (*angr.analyses.decompiler.optimization_passes.CrossJumpReverter attribute*), 543
STAGE (*angr.analyses.decompiler.optimization_passes.deadblock_remover.D attribute*), 565
STAGE (*angr.analyses.decompiler.optimization_passes.DeadblockRemover attribute*), 544
STAGE (*angr.analyses.decompiler.optimization_passes.determine_load_size attribute*), 565
STAGE (*angr.analyses.decompiler.optimization_passes.div_simplifier.DivSim attribute*), 566
STAGE (*angr.analyses.decompiler.optimization_passes.DivSimplifier attribute*), 544
STAGE (*angr.analyses.decompiler.optimization_passes.eager_std_string_con attribute*), 574
STAGE (*angr.analyses.decompiler.optimization_passes.eager_std_string_eva attribute*), 574
STAGE (*angr.analyses.decompiler.optimization_passes.EagerStdStringConca attribute*), 546
STAGE (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.Ex attribute*), 576
STAGE (*angr.analyses.decompiler.optimization_passes.ExprOpSwapper attribute*), 546
STAGE (*angr.analyses.decompiler.optimization_passes.flip_boolean_cmp.Fl attribute*), 576
STAGE (*angr.analyses.decompiler.optimization_passes.FlipBooleanCmp attribute*), 547
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_memcpy_sim attribute*), 577
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_memcpy_sim attribute*), 577
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_memset_simp attribute*), 578
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_memset_simp attribute*), 578
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_strcpy_simpl attribute*), 578
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_strcpy_simpl attribute*), 579
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_string_transf attribute*), 581
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_strlen_simpli attribute*), 581
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simp attribute*), 582
STAGE (*angr.analyses.decompiler.optimization_passes.inlined_wcscpy_simp attribute*), 582
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedMemcpySimpl attribute*), 548
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedMemcpySimpl attribute*), 548
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedMemsetSimpli attribute*), 548
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedMemsetSimpli attribute*), 548

STAGE (*angr.analyses.decompiler.optimization_passes.InlinedStringTransformationSimplifier attribute*), 549
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedStrcpySimplifier attribute*), 549
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedStrcpyConsolidation attribute*), 549
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedWcpcpySimplifier attribute*), 549
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedWcpcpyConsolidation attribute*), 550
STAGE (*angr.analyses.decompiler.optimization_passes.InlinedWcscpySimplifier attribute*), 550
STAGE (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter attribute*), 583
STAGE (*angr.analyses.decompiler.optimization_passes.ite_region_converter.ITERegionConverter attribute*), 583
STAGE (*angr.analyses.decompiler.optimization_passes.ITEExprConverter attribute*), 547
STAGE (*angr.analyses.decompiler.optimization_passes.ITERegionConverter attribute*), 547
STAGE (*angr.analyses.decompiler.optimization_passes.knowledge_base_optimizer.KnowledgeBaseOptimizer attribute*), 583
STAGE (*angr.analyses.decompiler.optimization_passes.KnowledgeBaseOptimizer attribute*), 550
STAGE (*angr.analyses.decompiler.optimization_passes.mips_gp_setting_simplifier.MipsGpSettingSimplifier attribute*), 585
STAGE (*angr.analyses.decompiler.optimization_passes.MipsGpSettingSimplifier attribute*), 552
STAGE (*angr.analyses.decompiler.optimization_passes.mod_simplifier.ModSimplifier attribute*), 586
STAGE (*angr.analyses.decompiler.optimization_passes.ModSimplifier attribute*), 552
STAGE (*angr.analyses.decompiler.optimization_passes.optimization_pass_stage.OptimizationPassStage attribute*), 589
STAGE (*angr.analyses.decompiler.optimization_passes.peephole_optimizations.PeepholeOptimizationExprBase attribute*), 590
STAGE (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier.RegisterSaveAreaSimplifier attribute*), 590
STAGE (*angr.analyses.decompiler.optimization_passes.register_save_area_simplifier_adv.RegisterSaveAreaSimplifierAdvanced attribute*), 591
STAGE (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifier attribute*), 553
STAGE (*angr.analyses.decompiler.optimization_passes.RegisterSaveAreaSimplifierAdvanced attribute*), 553
STAGE (*angr.analyses.decompiler.optimization_passes.ret_addr_save_simplifier.RetAddrSaveSimplifier attribute*), 591
STAGE (*angr.analyses.decompiler.optimization_passes.ret_deduplicator.ReturnDeduplicator attribute*), 591
STAGE (*angr.analyses.decompiler.optimization_passes.RetAddrSaveSimplifier attribute*), 553
STAGE (*angr.analyses.decompiler.optimization_passes.return_duplicator_high.ReturnDuplicatorHigh attribute*), 593
STAGE (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 554
STAGE (*angr.analyses.decompiler.optimization_passes.ReturnDuplicatorHigh attribute*), 554
STAGE (*angr.analyses.decompiler.optimization_passes.stack_canary_simplifier.StackCanarySimplifier attribute*), 594
STAGE (*angr.analyses.decompiler.optimization_passes.StackCanarySimplifier attribute*), 556
STAGE (*angr.analyses.decompiler.optimization_passes.static_vvar_rewriter.StaticVVarRewriter attribute*), 596
STAGE (*angr.analyses.decompiler.optimization_passes.StaticVVarRewriter attribute*), 556
STAGE (*angr.analyses.decompiler.optimization_passes.switch_default_case_lifter.SwitchDefaultCaseLifter attribute*), 596
STAGE (*angr.analyses.decompiler.optimization_passes.switch_reused_entry_rewriter.SwitchReusedEntryRewriter attribute*), 597
STAGE (*angr.analyses.decompiler.optimization_passes.SwitchDefaultCaseLifter attribute*), 557
STAGE (*angr.analyses.decompiler.optimization_passes.SwitchReusedEntryRewriter attribute*), 557
STAGE (*angr.analyses.decompiler.optimization_passes.tag_slicer.TagSlicer attribute*), 597
STAGE (*angr.analyses.decompiler.optimization_passes.TagSlicer attribute*), 558
STAGE (*angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier.WinStackCanarySimplifier attribute*), 598
STAGE (*angr.analyses.decompiler.optimization_passes.WinStackCanarySimplifier attribute*), 558
STAGE (*angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier.X86GccGetPcSimplifier attribute*), 598
STAGE (*angr.analyses.decompiler.optimization_passes.X86GccGetPcSimplifier attribute*), 558
STAGE (*angr.analyses.deobfuscator.string_obf_opt_passes.StringObfType3Rewriter attribute*), 739
STAGE (*angr.analyses.deobfuscator.string_obf_opt_passes.StringObfType3Rewriter attribute*), 733
STAGE (*angr.analyses.decompiler.cleanup_code_remover.CleanupCodeRemover attribute*), 1421
STAGE (*angr.analyses.decompiler.register_save_area_simplifier_rewriter.ComboRegisterSaveAreaSimplifier attribute*), 1421
STAGE (*angr.analyses.decompiler.optimization_passes.deref_coercion_simplifier.DerefCoercionSimplifier attribute*), 1422
STAGE (*angr.analyses.decompiler.optimization_passes.deref_coercion_simplifier_uninlined.DerefCoercionSimplifierUninlined attribute*), 1422
STAGE (*angr.analyses.decompiler.optimization_passes.error_propagation_simplifier.ErrorPropagationSimplifier attribute*), 1423
STAGE (*angr.analyses.decompiler.optimization_passes.function_prototype_inference.FunctionPrototypeInference attribute*), 1423
STAGE (*angr.analyses.decompiler.optimization_passes.macro.format_macro_simplifier.FormatMacroSimplifier attribute*), 1424
STAGE (*angr.analyses.decompiler.optimization_passes.outliners.string_cmp_outliner.StringCmpOutliner attribute*), 1424
STAGE (*angr.analyses.decompiler.optimization_passes.outliners.string_literal_outliner.StringLiteralOutliner attribute*), 1425
STAGE (*angr.analyses.decompiler.optimization_passes.outliners.string_outliner.StringOutliner attribute*), 1425

STAGE (*angr.rust.optimization_passes.outliners.unwrap_outliner.UnwrapOutliner attribute*), 1425
STAGE (*angr.rust.optimization_passes.outliners.vec_outliner.VecOutliner attribute*), 1426
STAGE (*angr.rust.optimization_passes.pattern_match_simplifier.PatternMatchSimplifier attribute*), 1426
STAGE (*angr.rust.optimization_passes.pre_pattern_match_simplifier.PrePatternMatchSimplifier attribute*), 1427
STAGE (*angr.rust.optimization_passes.redundant_block_remover.RedundantBlockRemover attribute*), 1428
STAGE (*angr.rust.optimization_passes.ret_expr_rewriter.RetExprRewriter attribute*), 1428
STAGE (*angr.rust.optimization_passes.rust_calling_convention.RustCallingConvention attribute*), 1428
STAGE (*angr.rust.optimization_passes.security_check_remover.SecurityCheckRemover attribute*), 1429
STAGE (*angr.rust.optimization_passes.str_argument_simplifier.StrArgumentSimplifier attribute*), 1429
STAGE (*angr.rust.optimization_passes.struct_instantiation_simplifier.StructInstantiationSimplifier attribute*), 1430
STAGE (*angr.rust.optimization_passes.struct_return_simplifier.StructReturnSimplifier attribute*), 1430
start (*angr.rustylib.Segment attribute*), 1439
start() (*angr.distributed.worker.Worker method*), 995
start() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
stash() (*angr.sim_manager.SimulationManager method*), 1445
stashes (*angr.sim_manager.SimulationManager property*), 1442
stat (*class in angr.procedures.linux_kernel.stat*), 1367
Stat (*class in angr.state_plugins.filesystem*), 1547
state (*angr.emulator.Emulator property*), 995
state (*angr.procedures.stubs.format_parser.FormatString property*), 1378
state (*angr.state_plugins.SimMemView attribute*), 1516
state (*angr.state_plugins.view.SimMemView attribute*), 1607
state_blank() (*angr.simos.cgc.SimCGC method*), 1486
state_blank() (*angr.simos.simos.SimOS method*), 1488
state_call() (*angr.simos.javavm.SimJavaVM method*), 1486
state_call() (*angr.simos.simos.SimOS method*), 1489
state_entry() (*angr.simos.cgc.SimCGC method*), 1486
state_entry() (*angr.simos.javavm.SimJavaVM method*), 1486
state_entry() (*angr.simos.linux.SimLinux method*), 1488
state_entry() (*angr.simos.simos.SimOS method*), 1489
state_entry() (*angr.simos.windows.SimWindows method*), 1492
state_full_init() (*angr.simos.linux.SimLinux method*), 1488
state_full_init() (*angr.simos.simos.SimOS method*), 1489
state_priority() (*angr.exploration_techniques.spiller.Spiller method*), 1465
StateDivergedException, 996
StateHierarchy (*class in angr.state_hierarchy*), 1494
STATEMENT (*angr.state_plugins.inspect.EventType attribute*), 1565
statement (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
statement (*angr.state_plugins.InspectAttrs attribute*), 1498
Statement (*class in angr.ailment.statement*), 225
statement_length() (*angr.storage.memory_mixins.regioned_memory.region_memory_mixin.RegionMemoryMixin property*), 1662
StatementLocationSimplifier (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 634
StatementLocationSimplifier.StatementLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 634
StatementLocation (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 634
statements (*angr.ailment.Block attribute*), 210
statements (*angr.ailment.block.Block attribute*), 215
statements (*angr.engines.pcode.lifter.IRSB property*), 1041
StateOption (*class in angr.sim_state_options*), 1456
StateType (*class in angr.analyses.purity.engine*), 802
StateType (*in module angr.engines.ail.engine_light*), 1003
Static (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 741
STATIC (*angr.simos.windows.SecurityCookieInit attribute*), 1492
static_exits() (*angr.sim_procedure.SimProcedure method*), 1451
StaticBuffers (*class in angr.protos.ail_types_pb2*), 1394
StaticBuffers.EntriesEntry (*class in angr.protos.ail_types_pb2*), 1394
StaticFindMixin (*class in angr.storage.memory_mixins.regioned_memory.static_find_mixin*), 1664
StaticHooker (*class in angr.analyses.static_hooker*), 896
StaticObjectFinder (*class in angr.analyses.find_objects_static*), 746
StaticVVar (*class in angr.protos.ail_types_pb2*), 1394
StaticVVarRewriter (*class in angr.analyses.decompiler.optimization_passes.static_vvar_rewriter*), 595
StaticVVars (*class in angr.protos.ail_types_pb2*), 1394

StaticVVars.EntriesEntry (*class in angr.protos.ail_types_pb2*), 1394
std____throw_bad_cast (*class in angr.procedures.libstdcpp.std____throw_bad_alloc*), 1363
std____throw_bad_cast (*class in angr.procedures.libstdcpp.std____throw_bad_cast*), 1363
std____throw_logic_error (*class in angr.procedures.libstdcpp.std____throw_length_error*), 1363
std____throw_logic_error (*class in angr.procedures.libstdcpp.std____throw_logic_error*), 1363
std__terminate (*class in angr.procedures.libstdcpp.std__terminate*), 1363
STD_SWAP (*in module angr.analyses.decompiler.known_patterns.std_swap*), 528
STD_VECTOR_EXACTDIV_SIZES (*in module angr.analyses.decompiler.known_patterns.std_vector_size*), 529
STD_VECTOR_STACK_TEMPLATES (*in module angr.analyses.decompiler.known_patterns.std_vector_size*), 529
StdinAtom (*class in angr.analyses.reaching_definitions.function_handler_library.stdio*), 846
StdoutAtom (*class in angr.analyses.reaching_definitions.function_handler_library.stdio*), 846
StdoutGuardMiddleware (*class in angr.mcp.server*), 1303
step() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1097
step() (*angr.exploration_techniques.director.Director method*), 1101
step() (*angr.sim_manager.SimulationManager method*), 1443
step() (*angr.sim_state.SimState method*), 1454
step_back() (*angr.analyses.reaching_definitions.call_trace.CallTrace method*), 834
step_state() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1097
step_state() (*angr.sim_manager.SimulationManager method*), 1444
stl_accessor (*angr.analyses.decompiler.structured_codegen.c.CVariableField attribute*), 680
stl_accessor_name() (*in module angr.analyses.decompiler.stl_field_accessors*), 661
stmt_anchor_key() (*in module angr.analyses.decompiler.known_patterns.dsl*), 506
stmt_can_move_to() (*angr.analyses.decompiler.optimization_passes.duplicate_reverter.duplicationReverter method*), 571
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.base.Pee attribute*), 603
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.base.Pee attribute*), 602
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.cas_intr attribute*), 607
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.cmpord_ attribute*), 609
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.coalesce attribute*), 609
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.remove_ attribute*), 614
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.remove_ attribute*), 614
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.rewrite_ attribute*), 619
stmt_classes (*angr.analyses.decompiler.peephole_optimizations.rol_ror.I attribute*), 619
stmt_classes (*angr.analyses.deobfuscator.string_obf_opt_passes.StringO attribute*), 739
stmt_classes (*angr.analyses.deobfuscator.StringObfType3Rewriter attribute*), 733
stmt_handlers (*angr.ailment.block_walker.AILBlockWalker property*), 215
stmt_idx (*angr.analyses.disassembly.SootStatement property*), 741
stmt_idx (*angr.errors.SimError attribute*), 1083
stmt_idx (*angr.SimError attribute*), 177
stmt_observe() (*angr.analyses.reaching_definitions.reaching_definitions. method*), 869
stmt_pattern_anchor_key() (*in module angr.analyses.decompiler.known_patterns.dsl*), 506
stmt_span (*angr.analyses.decompiler.known_patterns.finder.KnownPattern attribute*), 507
stmt_span (*angr.analyses.decompiler.known_patterns.KnownPatternMatch attribute*), 461
stmt_span (*angr.analyses.decompiler.KnownPatternMatch attribute*), 383
StmtDataType (*in module angr.analyses.purity.engine*), 803
stmts_used (*angr.engines.pcode.lifter.IRSB property*), 1040
StochasticSearch (*class in angr.exploration_techniques.stochastic*), 1106
STOP (*class in angr.state_plugins.unicorn_engine*), 1600
stop() (*angr.distributed.server.Server method*), 994
STOP_ERROR (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_EXECNONE (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_HLT (*angr.state_plugins.unicorn_engine.STOP.DuplicationReverter*

*tribute*), 1601
stop_message (*angr.state_plugins.unicorn_engine.STOP attribute*), 1602
STOP_NODECODE (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_NORMAL (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_NOSTART (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SEGFAULT (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_STOPPOINT (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYMBOLIC_BLOCK_EXIT_CONDITION (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYMBOLIC_BLOCK_EXIT_TARGET (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYMBOLIC_PC (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYMBOLIC_READ_ADDR (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYMBOLIC_READ_SYMBOLIC_TRACKING_DISABLED (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYMBOLIC_WRITE_ADDR (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYSCALL (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_SYSCALL_ARM (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNKNOWN_MEMORY_WRITE_SIZE (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_EXPR_GETI (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_EXPR_UNKNOWN (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_STMT_CAS (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_STMT_DIRTY (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_STMT_LLSC (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_STMT_LOADG (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_STMT_PUTI (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_STMT_STOREG (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_UNSUPPORTED_STMT_UNKNOWN (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_VEX_LIFT_FAILED (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_X86_CPUID (*angr.state_plugins.unicorn_engine.STOP attribute*), 1602
STOP_ZERO_DIV (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
STOP_ZEROPAGE (*angr.state_plugins.unicorn_engine.STOP attribute*), 1601
StopDetails (*class in angr.state_plugins.unicorn_engine*), 1602
stopped (*angr.distributed.server.Server property*), 994
storage (*angr.analyses.decompiler.edits.resolve.ResolvedVariable property*), 447
storage (*angr.state_plugins.sim_action.SimActionData property*), 1585
Store (*class in angr.ailment.statement*), 226
Store (*class in angr.analyses.typehoon.typevars*), 919
store() (*angr.analyses.stack_pointer_tracker.StackPointerTrackerState method*), 895
store() (*angr.sim_type.SimCppClass method*), 1475
store() (*angr.sim_type.SimStruct method*), 1471
store() (*angr.sim_type.SimType method*), 1460
store() (*angr.sim_type.SimTypeArray method*), 1468
store() (*angr.sim_type.SimTypeBitfield method*), 1475
store() (*angr.sim_type.SimTypeBool method*), 1465
store() (*angr.sim_type.SimTypeChar method*), 1464
store() (*angr.sim_type.SimTypeEnum method*), 1473
store() (*angr.sim_type.SimTypeFloat method*), 1470
store() (*angr.sim_type.SimTypeNum method*), 1463
store() (*angr.sim_type.SimTypeNumOffset method*), 1475
store() (*angr.sim_type.SimTypeReg method*), 1462
store() (*angr.sim_type.SimTypeWideChar method*), 1465
store() (*angr.sim_type.SimTypeWString method*), 1469
store() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 1546
store() (*angr.state_plugins.light_registers.SimLightRegisters method*), 1577
store() (*angr.state_plugins.view.SimMemView method*), 1607
store() (*angr.storage.memory_mixins.actions_mixin.ActionsMixinHigh method*), 1633

store() (*angr.storage.memory_mixins.actions_mixin.ActionsMixinLow method*), 1633
store() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 1634
store() (*angr.storage.memory_mixins.bvv_conversion_mixin.BVVConversionMixin method*), 1634
store() (*angr.storage.memory_mixins.conditional_store_mixin.ConditionalStoreMixin method*), 1634
store() (*angr.storage.memory_mixins.convenient_mappings_mixin.ConvenientMappingsMixin method*), 1634
store() (*angr.storage.memory_mixins.dirty_addrs_mixin.DirtyAddrsMixin method*), 1635
store() (*angr.storage.memory_mixins.inspect_mixin.InspectMixin method*), 1636
store() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1636
store() (*angr.storage.memory_mixins.keyvalue_memory_mixin.KeyValueMemoryMixin method*), 1638
store() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1639
store() (*angr.storage.memory_mixins.name_resolution_mixin.NameResolutionMixin method*), 1641
store() (*angr.storage.memory_mixins.paged_memory.pages_mixin.PagedMemoryMixin method*), 1642
store() (*angr.storage.memory_mixins.paged_memory.pages.history_tree.HistoryTreePage method*), 1648
store() (*angr.storage.memory_mixins.paged_memory.pages.multi_values.MultiValues method*), 1649
store() (*angr.storage.memory_mixins.paged_memory.pages.mv_list_page.MVListPage method*), 1651
store() (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage method*), 1655
store() (*angr.storage.memory_mixins.regioned_memory.region_meta_mixin.MemoryRegionMetaMixin method*), 1662
store() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 1663
store() (*angr.storage.memory_mixins.simple_interface_mixin.SimpleInterfaceMixin method*), 1664
store() (*angr.storage.memory_mixins.simplification_mixin.SimplificationMixin method*), 1665
store() (*angr.storage.memory_mixins.size_resolution_mixin.SizeConcretizationMixin method*), 1665
store() (*angr.storage.memory_mixins.size_resolution_mixin.SizeNormalizationMixin method*), 1665
store() (*angr.storage.memory_mixins.slotted_memory.SlottedMemoryMixin method*), 1665
store() (*angr.storage.memory_mixins.underconstrained_mixin.UnderconstrainedMixin method*), 1666
store() (*angr.storage.memory_mixins.unwrapper_mixin.UnwrapperMixin method*), 1666
store() (*angr.vaults.Vault method*), 1704
store() (*angr.vaults.VaultDirShelf method*), 1705
store_array_element() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1636
store_array_elements() (*angr.storage.memory_mixins.javavm_memory_mixin.JavaVmMemoryMixin method*), 1636
store_field() (*angr.engines.soot.values.thisref.SimSootValue_ThisRef method*), 1062
store_local_variable() (*angr.knowledge_plugins.propagations.states.PropagatorVEXState method*), 1275
store_register() (*angr.knowledge_plugins.propagations.states.PropagatorVEXState method*), 1276
store_tmp() (*angr.state_plugins.scratch.SimStateScratch method*), 1584
store_variable() (*angr.analyses.decompiler.optimization_passes.engine_base.SimEngineStructuring method*), 504
StoredObject (*class in angr.keyed_region*), 1115
StoreKeyValueMemoryMixin (*class in angr.storage.memory_mixins.keyvalue_memory_mixin*), 1638
StoreMemoryMixin (*class in angr.storage.memory_mixins.memory_mixin*), 1639
StoreNameResolutionMixin (*class in angr.storage.memory_mixins.name_resolution_mixin*), 1641
StoreSimplificationMixin (*class in angr.storage.memory_mixins.simplification_mixin*), 1665
StoreSizeConcretizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 1665
StoreSizeNormalizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 1665
StoreUnderconstrainedMixin (*class in angr.storage.memory_mixins.underconstrained_mixin*), 1666
StoreUnwrapperMixin (*class in angr.storage.memory_mixins.unwrapper_mixin*), 1666
StoredValue (*class in angr.keyed_region*), 1115
StoreMemoryMixin (*class in angr.storage.memory_mixins.memory_mixin*), 1639
StoreNameResolutionMixin (*class in angr.storage.memory_mixins.name_resolution_mixin*), 1641
StoreSimplificationMixin (*class in angr.storage.memory_mixins.simplification_mixin*), 1665
StoreSizeConcretizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 1665
StoreSizeNormalizationMixin (*class in angr.storage.memory_mixins.size_resolution_mixin*), 1665
StoreUnderconstrainedMixin (*class in angr.storage.memory_mixins.underconstrained_mixin*), 1666
StoreUnwrapperMixin (*class in angr.storage.memory_mixins.unwrapper_mixin*), 1666
StrcpyReplacer (*class in angr.analyses.identifier.functions.strcpy*), 782
strdup (*class in angr.procedures.posix.strdup*), 1375
StrictLangstonTracingMode (*in module angr.exploration_techniques.tracer*), 1107
STRICT_CALLER_SAVED_MATCH (*angr.calling_conventions.SimCC attribute*), 967
STRICT_CALLER_SAVED_MATCH (*angr.calling_conventions.SimCCGoAMD64 attribute*), 975
StrictLangstonTracingMode (*class in angr.exploration_techniques.tracer*), 1107
strlen (*class in angr.procedures.libc.strlen*), 1360
strncat (*class in angr.procedures.libc.strncat*), 1360
strncmp (*class in angr.procedures.libc.strncmp*), 1360
strncpy (*class in angr.procedures.libc.strncpy*), 1360
strnlen (*class in angr.procedures.libc.strnlen*), 1360
StrPtrArgumentSimplifier (*class in angr.rust.optimization_passes.str_argument_simplifier*), 1429
strstr (*class in angr.procedures.libc.strstr*), 1360
StructMemberFinder (*class in angr.analyses.decompiler.region_simplifiers.expr_folding*), 823
stpcpy (*class in angr.procedures.libc.stpcpy*), 1360
str_to_struct_ptr() (*angr.analyses.decompiler.structured_codegen.c.CConstant static method*), 682
str_to_struct_ptr() (*angr.analyses.decompiler.structured_codegen.rust.RustConstant static method*), 704
StrPtrArgumentSimplifier (*class in angr.rust.optimization_passes.str_argument_simplifier*), 1429
strcasecmp (*class in angr.analyses.identifier.functions.strcasecmp*), 780
strcasecmp (*class in angr.procedures.posix.strcasecmp*), 1376
strcat (*class in angr.procedures.libc.strcat*), 1360
strchr (*class in angr.procedures.libc.strchr*), 1360
strcmp (*class in angr.analyses.identifier.functions.strcmp*), 781
strcmp (*class in angr.procedures.libc.strcmp*), 1360
StrDerefSimplifierWalker (*class in angr.rust.optimization_passes.deref_coercion_simplifier*), 1428
strcoll (*class in angr.procedures.libc.strcoll*), 1360
strcpy (*class in angr.analyses.identifier.functions.strcpy*), 782
strcpy (*class in angr.procedures.libc.strcpy*), 1360
strdup (*class in angr.procedures.posix.strdup*), 1375

STRICT_CALLER_SAVED_MATCH (*angr.SimCC attribute*), 173
STRING (*angr.analyses.decompiler.structured_codegen.base.CConstantType attribute*), 668
String (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 798
String (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
String (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
STRING (*angr.llm_client.AgentKind attribute*), 1293
String (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
string (*angr.state_plugins.debug_variables.SimDebugVariable property*), 1545
string_capacity_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 517
string_data_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 517
string_escape() (*in module angr.analyses.reassembler*), 870
string_references() (*angr.knowledge_plugins.functions.function.Function method*), 1213
string_size_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 517
StringBuilderAppend (*class in angr.procedures.java_lang.stringbuilder*), 1349
StringBuilderInit (*class in angr.procedures.java_lang.stringbuilder*), 1349
StringCharAt (*class in angr.procedures.java_lang.string*), 1349
StringCmpOutliner (*class in angr.rust.optimization_passes.outliners.string_cmp_outliner*), 1424
StringConcat (*class in angr.procedures.java_lang.string*), 1349
StringDeobFuncDescriptor (*class in angr.analyses.deobfuscator.string_obf_finder*), 738
StringEquals (*class in angr.procedures.java_lang.string*), 1349
StringLength (*class in angr.procedures.java_lang.string*), 1349
StringLiteral (*class in angr.ailment.expression*), 222
StringLiteralOutliner (*class in angr.rust.optimization_passes.outliners.string_literal_outliner*), 1425
StringObfType1PeepholeOptimizer (*class in angr.analyses.deobfuscator.string_obf_peephole_optimizer*), 739
StringObfType3Rewriter (*class in angr.analyses.deobfuscator.string_obf_opt_passes*), 739
StringObfuscationFinder (*class in angr.analyses.deobfuscator.string_obf_finder*), 738
StringOutliner (*class in angr.rust.optimization_passes.outliners.string_outliner*), 1425
StringProxiNode (*class in angr.analyses.proximity_graph*), 799
StringSplit (*class in angr.procedures.java_lang.string*), 1349
StringTableSpec (*class in angr.tablespecs*), 1668
strip_simaction() (*in module angr.engines.vex.claripy.ccall*), 1068
strlen (*class in angr.analyses.identifier.functions.strlen*), 782
strlen (*class in angr.procedures.libc.strlen*), 1360
strncasecmp (*class in angr.procedures.posix.strncasecmp*), 1375
strncat (*class in angr.procedures.libc.strncat*), 1360
strncmp (*class in angr.analyses.identifier.functions.strncmp*), 783
strncmp (*class in angr.procedures.libc.strncmp*), 1360
strncpy (*class in angr.analyses.identifier.functions.strncpy*), 784
strncpy (*class in angr.procedures.libc.strncpy*), 1361
strndup (*class in angr.procedures.posix.strndup*), 1375
strnlen (*class in angr.procedures.libc.strnlen*), 1361
strstr (*class in angr.procedures.libc.strstr*), 1361
strtok (*class in angr.procedures.libc.strtok*), 1361
strtok_r (*class in angr.procedures.posix.strtok_r*), 1375
strtol (*class in angr.analyses.identifier.functions.strtol*), 784
strtol (*class in angr.procedures.libc.strtol*), 1361
strtol_inner() (*angr.procedures.libc.strtol.strtol static method*), 1361
strtoul (*class in angr.procedures.libc.strtoul*), 1361
struct (*angr.procedures.posix.readdir.readdir attribute*), 1374
struct (*angr.sim_type.SimStructValue property*), 1472
Struct (*class in angr.ailment.expression*), 222
Struct (*class in angr.analyses.typehoon.typeconsts*), 911
struct_name() (*angr.analyses.typehoon.translator.TypeTranslator method*), 906
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCCdecl attribute*), 972
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftAMD64 attribute*), 973

STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftCdecl attribute*), 972
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftFastcall attribute*), 972
STRUCT_RETURN_THRESHOLD (*angr.calling_conventions.SimCCMicrosoftThiscall attribute*), 972
StructBuilder (*class in angr.rust.analyses.struct_builder*), 1410
StructBuilder (*class in angr.rust.optimization_passes.struct_instantiation_simplifier*), 1429
StructInstantiationSimplifier (*class in angr.rust.optimization_passes.struct_instantiation_simplifier*), 1430
StructMatcher (*class in angr.rust.knowledge_plugins.known_structs*), 1412
StructMode (*class in angr.state_plugins.view*), 1607
StructReturnSimplifier (*class in angr.rust.optimization_passes.struct_return_simplifier*), 1430
STRUCTURED (*angr.llm_client.AgentKind attribute*), 1293
structured_node_has_multi_predecessors() (*in module angr.analyses.decompiler.utils*), 723
structured_node_is_simple_return() (*in module angr.analyses.decompiler.utils*), 723
structured_node_is_simple_return_strict() (*in module angr.analyses.decompiler.utils*), 724
StructuredCodeManager (*class in angr.knowledge_plugins.structured_code*), 1279
StructuredCodeManagerSerializer (*class in angr.angrdb.serializers.structured_code*), 954
structurer_class_from_name() (*in module angr.analyses.decompiler.structuring*), 716
StructurerBase (*class in angr.analyses.decompiler.structuring.structurer_base*), 719
STRUCTURING (*angr.analyses.decompiler.optimization_passes.const_prop_reverter.ConstPropOptReverter attribute*), 563
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ConstPropOptReverter attribute*), 542
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ctrl_ctrl_ctrl attribute*), 587
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ctrl_ctrl_ctrl attribute*), 589
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ctrl_ctrl_ctrl attribute*), 591
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ctrl_ctrl_ctrl attribute*), 593
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ReturnDuplicator attribute*), 554
STRUCTURING (*angr.analyses.decompiler.optimization_passes.ReturnDuplic attribute*), 554
STRUCTURING (*angr.rust.optimization_passes.pre_pattern_match_simplifier attribute*), 1427
StructuringOptimizationPass (*class in angr.analyses.decompiler.optimization_passes.optimization_pass*), 589
strxfrm (*class in angr.procedures.libc.strxfrm*), 1362
StubStasher (*class in angr.exploration_techniques.stub_stasher*), 1106
Sub (*angr.engines.light.ArithmeticExpression attribute*), 1005
Sub (*angr.engines.light.data.ArithmeticExpression attribute*), 1009
Sub (*class in angr.analyses.typehoon.typevars*), 916
sub_graph (*angr.analyses.data_dep.data_dependency_analysis.DataDepen property*), 358
sub_label() (*in module angr.analyses.typehoon.typevars*), 920
subgraph() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1186
subgraph() (*angr.knowledge_plugins.functions.function.Function method*), 1216
subgraph_between_nodes() (*in module angr.utils.graph*), 1683
subject (*angr.analyses.reaching_definitions.reaching_definitions.Reaching property*), 869
Subject (*class in angr.analyses.reaching_definitions.subject*), 869
SubjectType (*class in angr.analyses.reaching_definitions.subject*), 869
SubN (*class in angr.analyses.typehoon.typevars*), 919
subscribe_actions() (*angr.state_plugins.history.SimStateHistory method*), 1562
Subtype (*class in angr.analyses.typehoon.typevars*), 915
SUCCESS (*angr.analyses.cfg.indirect_jump_resolvers.mips_elf_fast.Case2Re attribute*), 338
successor_func() (*angr.analyses.cfg.AnnotatedCFG method*), 958
successor_node() (*angr.analyses.decompiler.region_overlay.RegionOve method*), 626
successors (*angr.analyses.decompiler.region_overlay.RegionOverlay property*), 627
successors (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
successors() (*angr.analyses.forward_analysis.visitors.graph.GraphVisito method*), 766
successors() (*angr.codeRoder.CodeNode method*), 987
successors() (*angr.exploration_techniques.base.ExplorationTechnique method*), 1098

successors() (*angr.factory.AngrObjectFactory method*), 1110
successors() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1185
successors() (*angr.sim_manager.SimulationManager method*), 1444
successors_and_jumpkinds() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 1174
SuccessorsEngine (*class in angr.engines.successors*), 1064
SuccessorsKind (*class in angr.analyses.pathfinder*), 794
Suggestions (*class in angr.exploration_techniques.suggestions*), 1106
SuperFastpathMixin (*class in angr.engines.vex.heavy.super_fastpath*), 1079
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.memory_mixin.MemoryMixin attribute*), 1638
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin attribute*), 1642
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.paged_memory.pages.ultra_page.UltraPage attribute*), 1654
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.paged_memory.pages.UltraPage attribute*), 1646
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.PagedMemoryMixin attribute*), 1625
SUPPORTS_CONCRETE_LOAD (*angr.storage.memory_mixins.UltraPage attribute*), 1631
SUPPORTS_OVERLAYS (*angr.analyses.decompiler.structuring.dream.DreamStructurer attribute*), 716
SUPPORTS_OVERLAYS (*angr.analyses.decompiler.structuring.structurer_base.SequenceStructurer attribute*), 714
SUPPORTS_OVERLAYS (*angr.analyses.decompiler.structuring.structurer_base.StructurerBase attribute*), 717
SUPPORTS_OVERLAYS (*angr.analyses.decompiler.structuring.sailr.SAILRStructurer attribute*), 715
supports_vector() (*in module angr.engines.vex.claripy.irop*), 1073
suppressed_by (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 458
suppressed_by (*angr.analyses.decompiler.known_patterns.known_pattern attribute*), 521
suppressed_by (*angr.analyses.decompiler.KnownPattern attribute*), 379
SV (*class in angr.analyses.fcp.fcp*), 744
switch_case_entry_node_has_common_successor_case_1() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer static method*), 718
switch_case_entry_node_has_common_successor_case_2() (*angr.analyses.decompiler.structuring.phoenix.PhoenixStructurer static method*), 718
switch_extract_bitwiseand_jumptable_info() (*in module angr.analyses.decompiler.utils*), 721
switch_extract_cmp_bounds() (*in module angr.analyses.decompiler.utils*), 720
switch_extract_cmp_bounds_from_condition() (*in module angr.analyses.decompiler.utils*), 720
switch_extract_switch_expr_from_jump_target() (*in module angr.analyses.decompiler.utils*), 721
SwitchBase (*class in angr.engines.soot.statements.switch*), 1055
SwitchCaseNode (*class in angr.analyses.decompiler.structurer_nodes*), 713
SwitchCaseRegion (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 639*
SwitchClusterFinder (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 639*
SwitchClusterReplacer (*class in angr.analyses.decompiler.region_simplifiers.switch_cluster_simpl 640*
SwitchDefaultCaseDuplicator (*class in angr.analyses.decompiler.optimization_passes.switch_default_cas 596*
SwitchExpressionSimplifier (*class in angr.analyses.decompiler.region_simplifiers.switch_expr_simplifi 641*
SwitchReusedEntryRewriter (*class in angr.analyses.decompiler.optimization_passes.switch_reused_ent 597*
symbol (*angr.analyses.reaching_definitions.function_handler.FunctionCall attribute*), 840
symbol (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
symbol (*angr.knowledge_plugins.functions.function.Function property*), 1214
symbol() (*in module angr.engines.vex.claripy.datalayer*), 1073
symbol_hooked_by() (*angr.project.Project method*), 1391
symbol_name (*angr.analyses.reassembler.NotypeLabel property*), 871
symbol_name (*angr.analyses.reassembler.ObjectLabel property*), 871
symbolic (*angr.engines.light.data.RegisterOffset property*), 1010
symbolic (*angr.knowledge_plugins.key_definitions.atoms.MemoryLocation property*), 1245

SYMBOLIC (*angr.simos.windows.SecurityCookieInit attribute*), 1492
symbolic (*angr.state_plugins.sim_action_object.SimActionObject property*), 1586
symbolic (*angr.storage.memory_object.SimMemoryObject property*), 1667
symbolic() (*angr.state_plugins.solver.SimSolver method*), 1595
symbolic_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
symbolic_expr (*angr.state_plugins.InspectAttrs attribute*), 1499
symbolic_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
symbolic_name (*angr.state_plugins.InspectAttrs attribute*), 1499
symbolic_size (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
symbolic_size (*angr.state_plugins.InspectAttrs attribute*), 1499
symbolic_stop_reasons (*angr.state_plugins.unicorn_engine.STOP attribute*), 1602
SYMBOLIC_VARIABLE (*angr.state_plugins.inspect.EventType attribute*), 1565
SymbolicBitmap (*class in angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap*), 1652
SymbolicMergerMixin (*class in angr.storage.memory_mixins.symbolic_merger_mixin*), 1666
symbolize() (*angr.analyses.reassembler.Reassembler method*), 877
symbolized (*angr.analyses.reassembler.Operand property*), 872
SymbolManager (*class in angr.analyses.reassembler*), 871
sympy_expr_to_claripy_ast() (*angr.analyses.decompiler.condition_processor.ConditionProcessor static method*), 410
Sys_int (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_int128 (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_int129 (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_int130 (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_int145 (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_int210 (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_int32 (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_syscall (*angr.protos.primitives_pb2.Edge attribute*), 1404
Sys_sysenter (*angr.protos.primitives_pb2.Edge attribute*), 1404
Syscall (*angr.protos.primitives_pb2.Edge attribute*), 1404
SYSCALL (*angr.state_plugins.inspect.EventType attribute*), 1565
syscall (*class in angr.procedures.stubs.syscall_stub*), 1380
syscall() (*angr.simos.simos.SimOS method*), 1489
syscall() (*angr.simos.userland.SimUserland method*), 1491
syscall() (*angr.simos.windows.SimWindows method*), 1492
syscall_abi() (*angr.simos.simos.SimOS method*), 1489
syscall_abi() (*angr.simos.userland.SimUserland method*), 1491
syscall_cc() (*angr.simos.simos.SimOS method*), 1490
syscall_cc() (*angr.simos.userland.SimUserland method*), 1491
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCN64LinuxSyscall attribute*), 981
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCO32LinuxSyscall attribute*), 980
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall attribute*), 982
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCPowerPCLinuxSyscall attribute*), 982
SYSCALL_ERRNO_START (*angr.calling_conventions.SimCCSyscall attribute*), 973
syscall_from_addr() (*angr.simos.simos.SimOS method*), 1490
syscall_from_addr() (*angr.simos.userland.SimUserland method*), 1491
syscall_from_addr() (*angr.simos.windows.SimWindows method*), 1493
syscall_from_number() (*angr.simos.simos.SimOS method*), 1490
syscall_from_number() (*angr.simos.userland.SimUserland method*), 1492
syscall_hook() (*angr.analyses.identifier.runner.Runner static method*), 785
syscall_hook() (*in module angr.state_plugins.trace_additions*), 1597

syscall_hook_concrete_rand() (*angr.analyses.identifier.runner.Runner static method*), 785
syscall_name (*angr.knowledge_plugins.cfg.cfg_node.CFGENode property*), 1175
syscall_name (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
syscall_name (*angr.state_plugins.InspectAttrs attribute*), 1499
syscall_num() (*angr.calling_conventions.SimCCAArch64LinuxSyscall static method*), 978
syscall_num() (*angr.calling_conventions.SimCCAArch64WindowsSyscall static method*), 979
syscall_num() (*angr.calling_conventions.SimCCAMD64LinuxSyscall static method*), 976
syscall_num() (*angr.calling_conventions.SimCCAMD64WindowsSyscall static method*), 976
syscall_num() (*angr.calling_conventions.SimCCARMLinuxSyscall static method*), 977
syscall_num() (*angr.calling_conventions.SimCCARMWindowsSyscall static method*), 978
syscall_num() (*angr.calling_conventions.SimCCN64LinuxSyscall static method*), 981
syscall_num() (*angr.calling_conventions.SimCCO32LinuxSyscall static method*), 980
syscall_num() (*angr.calling_conventions.SimCCPowerPC64LinuxSyscall static method*), 982
syscall_num() (*angr.calling_conventions.SimCCPowerPCLinuxSyscall static method*), 982
syscall_num() (*angr.calling_conventions.SimCCRISCV64LinuxSyscall static method*), 979
syscall_num() (*angr.calling_conventions.SimCCS390XLinuxSyscall static method*), 983
syscall_num() (*angr.calling_conventions.SimCCSyscall static method*), 973
syscall_num() (*angr.calling_conventions.SimCCX86LinuxSyscall static method*), 974
syscall_num() (*angr.calling_conventions.SimCCX86WindowsSyscall static method*), 974
SyscallNode (*class in angr.codenode*), 988
SyscallResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.syscall_resolver*), 340
sysinfo (*class in angr.procedures.linux_kernel.sysinfo*), 1367
syslog (*class in angr.procedures.posix.syslog*), 1376
system (*class in angr.procedures.libc.system*), 1362
SystemAtom (*class in angr.analyses.reaching_definitions.function_handler_library.stdlib*), 850
SystemCurrentTimeMillis (*class in angr.procedures.java_lang.system*), 1349

## T

tag (*angr.analyses.cfg.cfg_base.CFGBase attribute*), 307
tag (*angr.analyses.cfg.cfg_emulated.CFGEmulated attribute*), 311
tag (*angr.analyses.cfg.cfg_fast.CFGFast attribute*), 321
tag (*angr.analyses.cfg.CFGBase attribute*), 292
tag (*angr.analyses.cfg.CFGEmulated attribute*), 295
tag (*angr.analyses.cfg.CFGFast attribute*), 299
tag (*angr.analyses.CFGEmulated attribute*), 238
tag (*angr.analyses.CFGFast attribute*), 242
Tag (*class in angr.knowledge_plugins.key_definitions.tag*), 1201
tag_pool (*angr.analyses.decompiler.structured_codegen.c_serialize.SerialInfo property*), 688
TagDict (*class in angr.ailment.tagged_object*), 226
TaggedIntervalMap (*class in angr.utils.tagged_interval_map*), 1700
TaggedObject (*class in angr.ailment.tagged_object*), 227
tags (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 839
TagSlicer (*class in angr.analyses.decompiler.optimization_passes.tag_slicer*), 597
TAINT_ENTITY_MEM (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 1599
TAINT_ENTITY_NONE (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 1599
TAINT_ENTITY_REG (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 1599
TAINT_ENTITY_TMP (*angr.state_plugins.unicorn_engine.TaintEntityEnum attribute*), 1599
TaintEntityEnum (*class in angr.state_plugins.unicorn_engine*), 1599
tally() (*angr.sim_state_options.SimStateOptions method*), 1459
target (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 349
TargetGate (*class in angr.analyses.decompiler.known_patterns.gating*), 512
tc2simtype() (*angr.analyses.typehoon.translator.TypeTranslator method*), 906
teardown_callsite() (*angr.calling_conventions.SimCC method*), 970
TechniqueBuilder (*class in angr.exploration_techniques.tech_builder*), 1106
tell() (*angr.storage.file.SimFileDescriptorBase method*), 1614
TemporaryNode (*class in angr.utils.graph*), 1684
TemporaryVariable (*class in angr.protos.variables_pb2*), 1405
terminal_call() (*angr.rust.mixins.cfa_mixin.CFAMixin method*), 1417

terminate_execution() (*angr.engines.soot.engine.SootMixin static method*), 1047
terminate_execution() (*angr.project.Project method*), 1392
terminate_thread() (*angr.procedures.posix.pthread.pthread_exit method*), 1373
test() (*angr.analyses.identifier.runner.Runner method*), 785
test_empty_condition_node() (*angr.analyses.decompiler.structurer_nodes.BaseNode static method*), 711
test_empty_node() (*angr.analyses.decompiler.structurer_nodes.BaseNode static method*), 711
test_path() (*angr.analyses.pathfinder.Pathfinder method*), 795
test_reducibility() (*angr.analyses.decompiler.region_identifier.RegionIdentifier method*), 623
test_unsupported_overlap() (*angr.knowledge_plugins.debug_variables.DebugVariables method*), 1192
TestData (*class in angr.analyses.identifier.func*), 771
TestPathReport (*class in angr.analyses.pathfinder*), 794
text_stale (*angr.analyses.decompiler.edits.Refresh attribute*), 427
text_stale (*angr.analyses.decompiler.edits.results.Refresh attribute*), 449
text_stale_all (*angr.analyses.decompiler.edits.Refresh attribute*), 427
text_stale_all (*angr.analyses.decompiler.edits.results.Refresh attribute*), 449
tgkill (*class in angr.procedures.linux_kernel.tgkill*), 1367
TGraph (*in module angr.analyses.decompiler.region_identifier*), 623
Threading (*class in angr.exploration_techniques.threading*), 1107
ThreeOrFour (*class in angr.analyses.identifier.functions.int2str*), 773
THUMB (*angr.analyses.cfg.cfg_fast.ARMDecodingMode attribute*), 315
thumb (*angr.knowledge_plugins.cfg.cfg_node.CFGNode property*), 1174
thumb (*angr.sim_state.SimState property*), 1456
tib_ptr (*angr.procedures.ntdll.exceptions.KiUserExceptionDispatcher attribute*), 1369
tidy_data_references() (*angr.knowledge_plugins.cfg.cfg_model.CFGModel method*), 1172
TidyStackAddr (*class in angr.analyses.decompiler.peephole_optimizations.tidy_stack_addr*), 621
time (*class in angr.procedures.libc.time*), 1362
time (*class in angr.procedures.linux_kernel.time*), 1367
timed_function() (*in module angr.state_plugins.solver*), 1587
Timeout (*class in angr.exploration_techniques.timeout*), 1107
timestamp (*angr.procedures.win32.sim_time.GetSystemTimeAsFileTime attribute*), 1386
timethis() (*in module angr.utils.timing*), 1700
TlsAlloc (*class in angr.procedures.win32.local_storage*), 1385
TlsBaseResolver (*class in angr.procedures.linux_loader.tls*), 1368
TlsFree (*class in angr.procedures.win32.local_storage*), 1385
TlsGetValue (*class in angr.procedures.win32.local_storage*), 1385
TlsSetValue (*class in angr.procedures.win32.local_storage*), 1385
TManager (*in module angr.analyses.decompiler.region_identifier*), 623
Tmp (*angr.analyses.data_dep.dep_nodes.DepNodeTypes attribute*), 359
Tmp (*angr.analyses.data_dep.DepNodeTypes attribute*), 356
TMP (*angr.analyses.reaching_definitions.AtomKind attribute*), 806
TMP (*angr.knowledge_plugins.key_definitions.atoms.AtomKind attribute*), 1241
TMP (*angr.state_plugins.sim_action.SimAction attribute*), 1584
TMP (*angr.state_plugins.SimAction attribute*), 1503
Tmp (*class in angr.ailment.expression*), 222
Tmp (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 334
Tmp (*class in angr.knowledge_plugins.key_definitions.atoms*), 1244
tmp_deps (*angr.state_plugins.sim_action.SimAction property*), 1584
tmp_deps (*angr.state_plugins.sim_action.SimActionData property*), 1585
tmp_expr() (*angr.state_plugins.scratch.SimStateScratch method*), 1583
tmp_idx (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 1244
tmp_idx (*angr.knowledge_plugins.key_definitions.definition.DefinitionMat attribute*), 1246
TMP_READ (*angr.state_plugins.inspect.EventType attribute*), 1565
tmp_read_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
tmp_read_expr (*angr.state_plugins.InspectAttrs attribute*), 1498

tmp_read_num (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
tmp_read_num (*angr.state_plugins.InspectAttrs attribute*), 1498
tmp_uses (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
TMP_WRITE (*angr.state_plugins.inspect.EventType attribute*), 1565
tmp_write_expr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1567
tmp_write_expr (*angr.state_plugins.InspectAttrs attribute*), 1498
tmp_write_num (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
tmp_write_num (*angr.state_plugins.InspectAttrs attribute*), 1498
TmpDepNode (*class in angr.analyses.data_dep.dep_nodes*), 360
tmpnam (*class in angr.procedures.libc.tmpnam*), 1362
tmps (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState property*), 862
TmpUsesCollector (*class in angr.utils.ssa.tmp_uses_collector*), 1699
TNode (*in module angr.analyses.decompiler.region_identifier*), 623
to_acyclic() (*angr.analyses.decompiler.region_overlay.RegionOverlayGraph method*), 632
to_acyclic_by_order() (*angr.analyses.decompiler.region_overlay.RegionOverlayGraph method*), 632
to_acyclic_by_order() (*angr.utils.graph.DirectedGraphHelper method*), 1689
to_acyclic_graph() (*in module angr.utils.graph*), 1683
to_ail_supergraph() (*in module angr.analyses.decompiler.utils*), 721
to_bits (*angr.analyses.decompiler.known_patterns.dsl.PConv attribute*), 495
to_bits (*angr.analyses.decompiler.known_patterns.PConv attribute*), 470
to_bytes() (*angr.ailment.Block method*), 210
to_bytes() (*angr.ailment.block.Block method*), 215
to_claripy() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1586
to_codenode() (*angr.knowledge_plugins.cfg.cfg_node.CFGNode method*), 1174
to_json() (*angr.analyses.decompiler.notes.decompilation_note.DecompilationNote method*), 538
to_json() (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.OpDescriptor method*), 575
to_json() (*angr.analyses.decompiler.variable_map.VariableMap method*), 731
to_json() (*angr.angrdb.serializers.structured_code.ConstantsSerializer static method*), 954
to_json() (*angr.knowledge_plugins.functions.function.FunctionInfo method*), 1209
to_json() (*angr.knowledge_plugins.functions.function_parser.CallingConventionStub method*), 1227
to_json() (*angr.procedures.definitions.SimTypeCollection method*), 1324
to_json() (*angr.rust.sim_type.EnumVariant method*), 1434
to_json() (*angr.sim_type.SimType method*), 1460
to_jsonable() (*angr.analyses.decompiler.notes.decompilation_note.DecompilationNote method*), 538
to_jsonable() (*angr.analyses.decompiler.notes.deobfuscated_strings.DeobfuscatedStringsNote method*), 539
to_jsonable() (*angr.analyses.decompiler.optimization_passes.expr_op_swapper.OpDescriptor method*), 575
to_label() (*angr.knowledge_plugins.propagations.prop_value.PropValue method*), 1269
to_networkx() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFG method*), 1186
to_string() (*angr.knowledge_plugins.xrefs.xref_types.XRefType static method*), 1293
to_valueset() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionObject method*), 1660
toggle_bookmark() (*angr.knowledge_plugins.bookmarks.Bookmarks method*), 1150
tolower (*class in angr.procedures.libc.tolower*), 1362
TOLOWER_LOC_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 1576
TOLOWER_LOC_ARRAY (*angr.state_plugins.SimStateLibc attribute*), 1531
top (*angr.state_plugins.callstack.CallStack property*), 1543
Top (*class in angr.analyses.propagator.values*), 798
top() (*angr.analyses.reaching_definitions.rd_state.ReachingDefinitionsState method*), 860
top() (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryBase static method*), 928
top() (*angr.knowledge_plugins.key_definitions.live_definitions.LiveDefinitions static method*), 1251
top() (*angr.knowledge_plugins.propagations.states.PropagatorState static method*), 1273
top_record (*angr.procedures.ntdll.exceptions.KiUserExceptionDispatcher attribute*), 1369
TopMergerMixin (*class in angr.storage.memory_mixins.top_merger_mixin*), 1666
TopType (*class in angr.analyses.typehoon.typeconsts*), 907
total_count() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeLocation property*), 1180
total_count (*angr.knowledge_plugins.functions.function_manager.SpillingFunctionManager property*), 1220
total_function_count

(*angr.knowledge_plugins.functions.function_manager.FunctionManager property*), 1227
total_size (*angr.analyses.cfg.pe_msvc_eh_structs.EH4ScopeTable property*), 344
toupper (*class in angr.procedures.libc.toupper*), 1362
TOUPPER_LOC_ARRAY (*angr.state_plugins.libc.SimStateLibc attribute*), 1576
TOUPPER_LOC_ARRAY (*angr.state_plugins.SimStateLibc attribute*), 1531
TOverlay (*in module angr.analyses.decompiler.region_identifier*), 623
TraceActions (*class in angr.analyses.smc*), 891
TraceClassifier (*class in angr.analyses.smc*), 891
Tracer (*class in angr.exploration_techniques.tracer*), 1108
TracerDesyncError, 1107
TracerEnvironmentError, 1083
TracingMode (*class in angr.exploration_techniques.tracer*), 1107
TRACK_STACK_POINTERS (*angr.analyses.decompiler.clinic.ClinicStage attribute*), 404
TrackActionsMixin (*class in angr.engines.vex.heavy.actions*), 1075
transfer() (*angr.analyses.decompiler.variable_map.VariableMap method*), 731
transform_graph_from_ssa() (*angr.analyses.decompiler.decompiler.Decompiler method*), 416
transform_seqnode_from_ssa() (*angr.analyses.decompiler.decompiler.Decompiler method*), 416
transition_graph_ex() (*angr.knowledge_plugins.functions.function.Function method*), 1216
transitions (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 349
transitions_as_tuples (*angr.analyses.cfg_slice_to_sink.cfg_slice_to_sink.CFGSliceToSink property*), 349
transitive_closure() (*angr.analyses.reaching_definitions.dep_graph.DepGraph method*), 835
translate() (*angr.calling_conventions.AllocHelper method*), 963
translate_array_index() (*angr.engines.soot.values.arrayref.SimSootValue_ArrayRef static method*), 1059
translate_expr() (*in module angr.engines.soot.expressions*), 1049
translate_rm() (*in module angr.engines.vex.heavy.concretizers*), 1075
translate_stmt() (*in module angr.engines.soot.statements*), 1053
translate_value() (*in module angr.engines.soot.values*), 1058
translit (*class in angr.procedures.cgc.transmit*), 1324
transmit (*class in angr.procedures.tracer.transmit*), 1381
TRANSMIT_RECORD (*class in angr.state_plugins.unicorn_engine*), 1599
TraversalAnalysis (*class in angr.analyses.decompiler.ssailification.traversal*), 655
TraversalState (*class in angr.analyses.decompiler.ssailification.traversal_state*), 658
traverse_in_order() (*in module angr.analyses.decompiler.ailgraph_walker*), 398
TreeIter (*class in angr.state_plugins.history*), 1563
trim() (*angr.state_plugins.history.SimStateHistory method*), 1561
trimmed() (*angr.analyses.cfg.cfb.Unknown method*), 304
triop_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1012
truncate_bits() (*in module angr.utils.bits*), 1674
TrueMap (*angr.analyses.cfg.indirect_jump_resolvers.jumptable.AddressTransformation attribute*), 334
try_convert_to_enum() (*angr.rust.optimization_passes.struct_return_simplifier.StructRetu method*), 1430
try_match() (*angr.analyses.identifier.functions.free.free method*), 773
try_str_literal() (*angr.rust.optimization_passes.str_argument_simplif method*), 1429
try_str_reference() (*angr.rust.optimization_passes.str_argument_simplifier.StrArgum method*), 1429
try_unpack_const() (*angr.engines.light.data.ArithmeticExpression static method*), 1010
TryBlockEntry (*class in angr.analyses.cfg.pe_msvc_eh_structs*), 344
TwoOrThree (*class in angr.analyses.identifier.functions.int2str*), 773
Tx (*in module angr.analyses.decompiler.region_overlay*), 624
ty (*angr.sim_type.TypeRef property*), 1461
tyenv (*angr.ailment.Manager attribute*), 212
tyenv (*angr.ailment.manager.Manager attribute*), 224
tyenv (*angr.engines.pcode.lifter.IRSB property*), 1040
type (*angr.analyses.data_dep.dep_nodes.BaseDepNode property*), 359
type (*angr.analyses.decompiler.known_patterns.pattern.PatternParam attribute*), 520
type (*angr.analyses.decompiler.known_patterns.PatternParam attribute*), 480

type (*angr.analyses.decompiler.structured_codegen.c.CBinaryOp property*), 681
type (*angr.analyses.decompiler.structured_codegen.c.CConstant property*), 682
type (*angr.analyses.decompiler.structured_codegen.c.CDirtyExpression property*), 684
type (*angr.analyses.decompiler.structured_codegen.c.CDirtyStatement property*), 678
type (*angr.analyses.decompiler.structured_codegen.c.CExpression property*), 674
type (*angr.analyses.decompiler.structured_codegen.c.CFakeVariable property*), 679
type (*angr.analyses.decompiler.structured_codegen.c.CFunctionCall property*), 677
type (*angr.analyses.decompiler.structured_codegen.c.CIndexedVariable property*), 680
type (*angr.analyses.decompiler.structured_codegen.c.CITE property*), 682
type (*angr.analyses.decompiler.structured_codegen.c.CMultiStatementExpression property*), 683
type (*angr.analyses.decompiler.structured_codegen.c.CRegister property*), 682
type (*angr.analyses.decompiler.structured_codegen.c.CStructField property*), 679
type (*angr.analyses.decompiler.structured_codegen.c.CTypeCast property*), 681
type (*angr.analyses.decompiler.structured_codegen.c.CUnaryOp property*), 680
type (*angr.analyses.decompiler.structured_codegen.c.CVariable property*), 679
type (*angr.analyses.decompiler.structured_codegen.c.CVariableField property*), 680
type (*angr.analyses.decompiler.structured_codegen.c.CVectorConvert property*), 683
type (*angr.analyses.decompiler.structured_codegen.c.CVEXCCallExpression property*), 683
type (*angr.analyses.decompiler.structured_codegen.rust.RustBreak property*), 703
type (*angr.analyses.decompiler.structured_codegen.rust.RustEnum property*), 704
type (*angr.analyses.decompiler.structured_codegen.rust.RustEnumField property*), 706
type (*angr.analyses.decompiler.structured_codegen.rust.RustExpression property*), 693
type (*angr.analyses.decompiler.structured_codegen.rust.RustFieldAccess property*), 701
type (*angr.analyses.decompiler.structured_codegen.rust.RustFunctionCall property*), 698
type (*angr.analyses.decompiler.structured_codegen.rust.RustFunctionLikeMacro property*), 697
type (*angr.analyses.decompiler.structured_codegen.rust.RustFunctionTable property*), 702
type (*angr.analyses.decompiler.structured_codegen.rust.RustImport property*), 705
type (*angr.analyses.decompiler.structured_codegen.rust.RustLet property*), 700
type (*angr.analyses.decompiler.structured_codegen.rust.RustMultiStatement property*), 705
type (*angr.analyses.decompiler.structured_codegen.rust.RustRegister property*), 704
type (*angr.analyses.decompiler.structured_codegen.rust.RustStringLiteral property*), 701
type (*angr.analyses.decompiler.structured_codegen.rust.RustStructField property*), 700
type (*angr.analyses.decompiler.structured_codegen.rust.RustTypeCast property*), 703
type (*angr.analyses.decompiler.structured_codegen.rust.RustUnaryOp property*), 702
type (*angr.analyses.decompiler.structured_codegen.rust.RustVariable property*), 701
type (*angr.analyses.decompiler.structured_codegen.rust.RustVariableField property*), 702
type (*angr.analyses.decompiler.structured_codegen.rust.RustVectorConvert property*), 706
type (*angr.analyses.decompiler.structured_codegen.rust.RustVEXCCallExp property*), 705
type (*angr.analyses.reaching_definitions.subject.Subject property*), 870
type (*angr.rust.sim_type.EnumVariant property*), 1434
type (*angr.sim_type.TypeRef property*), 1461
TypeAssignmentFinder (*class in angr.analyses.deobfuscator.api_obf_finder*), 734
TYPE_1 (*angr.analyses.deobfuscator.api_obf_finder.APIObfuscationType attribute*), 734
type_collections_for_lib() (*in module angr.utils.types*), 1702
type_equals() (*in module angr.analyses.decompiler.structured_codegen.c*), 671
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Array attribute*), 911
TYPE_HASH (*angr.analyses.typehoon.typeconsts.BottomType attribute*), 907
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Enum attribute*), 912
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Fd attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Float attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Float32 attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Float64 attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Function attribute*), 912
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int attribute*), 907

TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int1 attribute*), 907
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int128 attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int16 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int256 attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int32 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int512 attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int64 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Int8 attribute*), 907
TYPE_HASH (*angr.analyses.typehoon.typeconsts.IntVar attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Pointer attribute*), 910
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Pointer32 attribute*), 910
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Pointer64 attribute*), 910
TYPE_HASH (*angr.analyses.typehoon.typeconsts.RustEnum attribute*), 911
TYPE_HASH (*angr.analyses.typehoon.typeconsts.SInt16 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.SInt32 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.SInt64 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.SInt8 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.Struct attribute*), 911
TYPE_HASH (*angr.analyses.typehoon.typeconsts.TopType attribute*), 907
TYPE_HASH (*angr.analyses.typehoon.typeconsts.TypeConstant attribute*), 907
TYPE_HASH (*angr.analyses.typehoon.typeconsts.TypeVariableReference attribute*), 913
TYPE_HASH (*angr.analyses.typehoon.typeconsts.UInt16 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.UInt32 attribute*), 908
TYPE_HASH (*angr.analyses.typehoon.typeconsts.UInt64 attribute*), 909
TYPE_HASH (*angr.analyses.typehoon.typeconsts.UInt8 attribute*), 908
type_hint (*angr.analyses.decompiler.known_patterns.PatternParam attribute*), 520
type_hint (*angr.analyses.decompiler.known_patterns.WildcardPattern attribute*), 480
type_layout_key() (*in module angr.analyses.decompiler.structured_codegen.c*), 671
type_parser_singleton() (*in module angr.sim_type*), 1476
type_pool (*angr.analyses.decompiler.structured_codegen.c_serialize.SerialStructuredCodeGenerator property*), 688
type_string (*angr.knowledge_plugins.xrefs.xref.XRef property*), 1292
type_to_c_repr_chunks() (*in module angr.analyses.decompiler.structured_codegen.c*), 671
type_to_rust_repr_chunks() (*in module angr.analyses.decompiler.structured_codegen.rust*), 692
TypeConstant (*class in angr.analyses.typehoon.typeconsts*), 906
TypeConstraint (*class in angr.analyses.typehoon.typevars*), 914
TypeDBLoader (*class in angr.rust.analyses.type_db_loader*), 1411
TypedVariable (*class in angr.storage.memory_mixins.keyvalue_memory_mixin*), 1637
TypeHints (*class in angr.rust.knowledge_plugins.type_hints*), 1413
Typehoon (*class in angr.analyses.typehoon.typehoon*), 913
TypeLattice (*class in angr.analyses.typehoon.simple_solver*), 899
TypeParseError, 438
TypeRef (*class in angr.sim_type*), 1461
types (*angr.state_plugins.SimMemView attribute*), 1514
types (*angr.state_plugins.view.SimMemView attribute*), 1605
TypesStore (*class in angr.knowledge_plugins.types*), 1280
TypeTranslator (*class in angr.analyses.typehoon.translator*), 905
typevar_to_variable() (*angr.analyses.typehoon.typevars.TypeVariables method*), 919
TypeVariable (*class in angr.analyses.typehoon.typevars*), 917
TypeVariableManager (*class in angr.analyses.typehoon.typevars*), 916
TypeVariableReference (*class in angr.analyses.typehoon.typeconsts*), 913
TypeVariables (*class in angr.analyses.typehoon.typevars*), 918
tzset() (*in module angr.procedures.posix.tz*), 1376

## U

u2s() (*in module angr.utils.bits*), 1674

u64_to_u32() (*in module angr.engines.vex.claripy.ccall*), 1073
UberEngine (*class in angr.engines*), 1000
UberEnginePcode (*class in angr.engines*), 1001
UberIcicleEngine (*class in angr.engines.icicle*), 1005
uc (*angr.state_plugins.unicorn_engine.Unicorn property*), 1603
UC_CONFIG (*angr.state_plugins.Unicorn attribute*), 1540
UC_CONFIG (*angr.state_plugins.unicorn_engine.Unicorn attribute*), 1603
UDef (*in module angr.analyses.decompiler.ssailification.ssailification*), 655
ugetrlimit (*class in angr.procedures.linux_kernel.getrlimit*), 1366
UInt16 (*class in angr.analyses.typehoon.typeconsts*), 908
UInt32 (*class in angr.analyses.typehoon.typeconsts*), 908
UInt64 (*class in angr.analyses.typehoon.typeconsts*), 908
UInt8 (*class in angr.analyses.typehoon.typeconsts*), 908
UltraPage (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 1654
UltraPagesMixin (*class in angr.storage.memory_mixins.paged_memory.paged_memory_mixin*), 1643
uname (*class in angr.procedures.linux_kernel.uname*), 1368
UnaryOp (*class in angr.ailment.expression*), 222
uncache_region() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
Unconstrained() (*angr.state_plugins.solver.SimSolver method*), 1588
UnconstrainedMethod (*class in angr.procedures.java.unconstrained*), 1337
Undefined (*class in angr.knowledge_plugins.key_definitions.undefined*), 1262
UnderconstrainedMixin (*class in angr.storage.memory_mixins.underconstrained_mixin*), 1666
underlying_edge_pairs() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 628
underlying_nodes() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 626
unfreeze() (*angr.analyses.stack_pointer_tracker.FrozenStackPointerTracker method*), 894
ungetc (*class in angr.procedures.libc.ungetc*), 1362
unhook() (*angr.project.Project method*), 1390
unhook_symbol() (*angr.project.Project method*), 1391
UnicodeString (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
UnicodeString (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
UnicodeString (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
Unicorn (*class in angr.state_plugins.unicorn_engine*), 1603
unified_variable() (*angr.knowledge_plugins.variables.variable_manag method*), 1288
uniform_value (*angr.storage.memory_mixins.paged_memory.pages.symb property*), 1652
unify_arch_name() (*in module angr.calling_conventions*), 984
unify_variables() (*angr.knowledge_plugins.variables.variable_manage method*), 1288
uninit_read_base (*angr.analyses.cfg.indirect_jump_resolvers.jumptable. attribute*), 334
UninitReadMeta (*class in angr.analyses.cfg.indirect_jump_resolvers.jumptable*), 333
union() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1586
unique() (*angr.state_plugins.solver.SimSolver method*), 1595
unique_type_name() (*angr.knowledge_plugins.types.TypesStore method*), 1280
UniqueSearch (*class in angr.exploration_techniques.unique*), 1109
Uniwrapper (*class in angr.state_plugins.unicorn_engine*), 1603
UNKNOWN (*angr.analyses.decompiler.ail_simplifier.DefEqRelation attribute*), 397
UNKNOWN (*angr.analyses.decompiler.stack_item.StackItemType attribute*), 659
Unknown (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 799
UNKNOWN (*angr.analyses.typehoon.simple_solver.ConstraintGraphTag attribute*), 901
Unknown (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 1175
Unknown (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 1163
Unknown (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
Unknown (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
UNKNOWN (*angr.sim_type.PointerDisposition attribute*), 1466
UnknownFunction (*class in angr.analyses.cfg.cfb*), 304
unknown_returning_func_addrs() (*angr.knowledge_plugins.functions.function_manager.FunctionM method*), 1224
UnknownDataType (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
UnknownJumpkind (*angr.protos.primitives_pb2.Edge attribute*), 1404
UnknownPatternError, 524
UnknownProxiNode (*class in angr.analyses.proximity_graph*), 800
UnknownSize (*class in*

*angr.knowledge_plugins.key_definitions.unknown_size*), 1262
UnknownSizeTag (*class in angr.knowledge_plugins.key_definitions.tag*), 1261
unlink (*class in angr.procedures.linux_kernel.unlink*), 1368
unlink (*class in angr.procedures.posix.unlink*), 1376
unlinks (*angr.state_plugins.filesystem.SimFilesystem property*), 1549
unmap_by_address() (*angr.storage.memory_mixins.regioned_memory.region_data.RegionMap method*), 1661
unmap_region() (*angr.storage.memory_mixins.address_concretization_mixin.AddressConcretizationMixin method*), 1634
unmap_region() (*angr.storage.memory_mixins.memory_mixin.MemoryMixin method*), 1639
unmap_region() (*angr.storage.memory_mixins.paged_memory.paged_memory_mixin.PagedMemoryMixin method*), 1642
unmatched_blocks (*angr.analyses.bindiff.FunctionDiff property*), 281
unmatched_functions (*angr.analyses.bindiff.BinDiff property*), 282
UnmatchedStatementsException, 280
unmount() (*angr.state_plugins.filesystem.SimFilesystem method*), 1549
unop_handler() (*angr.engines.light.engine.SimEngineLightVEX static method*), 1012
unpack_array() (*in module angr.analyses.decompiler.structured_codegen.rust*), 691
unpack_array() (*in module angr.utils.types*), 1701
unpack_pointer() (*in module angr.analyses.decompiler.structured_codegen.rust*), 691
unpack_pointer() (*in module angr.utils.types*), 1701
unpack_pointer_and_array() (*in module angr.utils.types*), 1701
unpack_typeref() (*in module angr.analyses.decompiler.structured_codegen.rust*), 691
unpack_typeref() (*in module angr.utils.types*), 1701
unqualified_name() (*angr.sim_type.NamedTypeMixin method*), 1462
Unreachable, 794
UNREACHABLE (*angr.analyses.codecave.CodeCaveClassification attribute*), 352
unreachable_history() (*angr.state_hierarchy.StateHierarchy method*), 1494
unreachable_state() (*angr.state_hierarchy.StateHierarchy method*), 1494
UnresolvableCallTarget (*class in angr.procedures.stubs.UnresolvableCallTarget*), 1377
UnresolvableJumpTarget (*class in angr.procedures.stubs.UnresolvableJumpTarget*), 1377
unresolvables (*angr.analyses.cfg.cfg_emulated.CFGEmulated property*), 314
unresolved_indirect_jumps (*angr.knowledge_base.KnowledgeBase property*), 1119
unroll_loops() (*angr.analyses.cfg.cfg_emulated.CFGEmulated method*), 312
UNSAT (*angr.analyses.pathfinder.SuccessorsKind attribute*), 794
unsat_core() (*angr.state_plugins.solver.SimSolver method*), 1591
UNSET (*angr.misc.telemetry.StatusCode attribute*), 1321
unset_stack_address_mapping() (*angr.storage.memory_mixins.regioned_memory.regioned_memory_mixin.RegionedMemoryMixin method*), 1664
unsigned_int_type() (*in module angr.analyses.typehoon.typeconsts*), 913
UnsignedExtension (*angr.analyses.cfg.indirect_jump_resolvers.jumptable attribute*), 334
Unspecified (*angr.knowledge_plugins.cfg.memory_data.MemoryDataSort attribute*), 1177
Unspecified (*angr.knowledge_plugins.cfg.MemoryDataSort attribute*), 1164
Unspecified (*angr.protos.cfg_pb2.MemoryData attribute*), 1396
unstash() (*angr.sim_manager.SimulationManager method*), 1446
unsupported_reasons (*angr.state_plugins.unicorn_engine.STOP attribute*), 1602
UnsupportedAILNodeError, 572
UnsupportedCCallError, 1085
UnsupportedDirtyError, 1085
UnsupportedEditError, 438
UnsupportedIRExprError, 1084
UnsupportedIROpError, 1084
UnsupportedIRStmtError, 1085
UnsupportedJNIFunction (*class in angr.procedures.java_jni.not_implemented*), 1346
UnsupportedNodeTypeError, 1087
UnsupportedOutlineError, 507
UnsupportedSyscallError (*in module angr.errors*), 1085
UnwindMapEntry (*class in angr.analyses.cfg.pe_msvc_eh_structs*), 342
unwrap_combo_reg_vvar_reference() (*in module angr.rust.utils.ail*), 1438
unwrap_conv() (*angr.analyses.decompiler.optimization_passes.return_dup static method*), 592

unwrap_stack_vvar_reference() (*in module angr.rust.utils.ail*), 1438
unwrap_stack_vvar_reference_with_offset() (*in module angr.rust.utils.ail*), 1438
UnwrapOutliner (*class in angr.rust.optimization_passes.outliners.unwrap_outliner*), 1425
UnwrapperMixin (*class in angr.storage.memory_mixins.unwrapper_mixin*), 1666
UnwrapSimplifierState (*class in angr.rust.optimization_passes.outliners.unwrap_outliner*), 1425
update() (*angr.analyses.purity.engine.ResultType method*), 802
update() (*angr.knowledge_plugins.propagations.propagation_model.PropagationModel method*), 1271
update() (*angr.procedures.definitions.SimLibrary method*), 1325
update() (*angr.rustylib.SegmentList method*), 1440
update() (*angr.sim_state_options.SimStateOptions method*), 1458
update() (*angr.storage.memory_mixins.regioned_memory.region_meta.RegionMeta method*), 1662
update_after_block_edits() (*angr.analyses.s_reaching_definitions.s_rda_model.SRDAModel method*), 886
update_block_indexes() (*angr.rust.mixins.cfg_transformation_mixin.CFGTransformationMixin method*), 1418
update_dbinfo() (*angr.angrdb.db.AngrDB method*), 941
update_func_block_count() (*angr.knowledge_plugins.functions.function.Function method*), 1214
update_graph_with_super_edits() (*angr.analyses.decompiler.optimization_passes.code_motion.CodeMotionOptimization static method*), 561
update_labels() (*in module angr.analyses.decompiler.utils*), 723
update_resolved_addrs() (*angr.knowledge_plugins.indirect_jumps.IndirectJumps method*), 1228
update_stackoff_vvars() (*angr.analyses.decompiler.stackarg_offset_manager.StackArgOffsetManager method*), 660
update_switch_case_list() (*in module angr.analyses.decompiler.region_simplifiers.switch_cluster_simplifier*), 640
update_variable_types() (*angr.analyses.typehoon.typehoon.Typehoon method*), 914
update_variable_types() (*angr.rust.typehoon.typehoon.RustTypehoon method*), 1437
UpdateArgumentsOption (*class in angr.analyses.calling_convention.calling_convention*), 285
UpdateWhenCCHasNoArgs (*angr.analyses.calling_convention.calling_convention.UpdateArg attribute*), 285
use_plugin_preset() (*angr.misc.plugins.PluginHub method*), 1318
use_technique() (*angr.sim_manager.SimulationManager method*), 1442
USER() (*angr.knowledge_plugins.functions.function.PrototypeSource attribute*), 1209
USER (*angr.knowledge_plugins.functions.PrototypeSource attribute*), 1208
UsercallArgSessionManager (*class in angr.calling_conventions*), 966
UserHook (*class in angr.procedures.stubs.UserHook*), 1378
Uses (*class in angr.knowledge_plugins.key_definitions.uses*), 1262
uses_by_codeloc (*angr.analyses.reaching_definitions.rd_state.ReachingD property*), 862
usleep (*class in angr.procedures.posix.usleep*), 1376

## V

va_arg() (*angr.sim_procedure.SimProcedure method*), 1451
validate() (*angr.sim_type.SimTypeBitfield method*), 1474
validate_name() (*in module angr.analyses.decompiler.edits.resolve*), 446
value (*angr.ailment.expression.ConvertType attribute*), 219
value (*angr.ailment.expression.VirtualVariableCategory attribute*), 223
value (*angr.ailment.ExpressionKind attribute*), 212
value (*angr.analyses.decompiler.known_patterns.dsl.PConst attribute*), 493
value (*angr.analyses.decompiler.known_patterns.PConst attribute*), 469
value (*angr.analyses.reaching_definitions.function_handler.FunctionEffect attribute*), 839
value (*angr.knowledge_plugins.key_definitions.heap_address.HeapAddress property*), 1248
Value (*class in angr.analyses.disassembly*), 742
Value (*in module angr.analyses.decompiler.ssailification.traversal_state*), 658
value() (*in module angr.engines.vex.claripy.datalayer*), 1073
value_and_labels() (*angr.knowledge_plugins.propagations.prop_value. method*), 1269
value_tuple() (*angr.analyses.data_dep.dep_nodes.BaseDepNode method*), 359

values() (*angr.analyses.decompiler.structured_codegen.base.SimTypeBottom method*), 669
values() (*angr.analyses.decompiler.structured_codegen.base.SimTypeChar method*), 668
values() (*angr.knowledge_plugins.cfg.spilling_cfg.SpillingCFGNodeData method*), 1180
values() (*angr.knowledge_plugins.patches.PatchManager method*), 1265
values() (*angr.state_plugins.globals.SimStateGlobals method*), 1551
values() (*angr.storage.memory_mixins.paged_memory.page_mixins.multi_values.MultiValues method*), 1650
Var2Unified (*class in angr.protos.variables_pb2*), 1405
var_args() (*angr.analyses.identifier.func.Func method*), 771
var_args() (*angr.analyses.identifier.functions.fdprintf.fdprintf method*), 773
var_args() (*angr.analyses.identifier.functions.printf.printf method*), 777
var_args() (*angr.analyses.identifier.functions.snprintf.snprintf method*), 780
var_args() (*angr.analyses.identifier.functions.sprintf.sprintf method*), 780
VarDepNode (*class in angr.analyses.data_dep.dep_nodes*), 360
Variable (*angr.analyses.proximity_graph.ProxiNodeTypes attribute*), 799
variable (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 1245
variable() (*angr.analyses.decompiler.variable_map.VariableMap method*), 727
variable_endoffset (*angr.analyses.decompiler.ssailification.traversal_engine.DefInfo property*), 656
variable_key_prefix (*angr.storage.memory_mixins.memory_mixin.MemoryMixin property*), 1638
variable_list_repr_chunks() (*angr.analyses.decompiler.structured_codegen.c.CFunction method*), 673
variable_list_repr_chunks() (*angr.analyses.decompiler.structured_codegen.rust.RustFunction method*), 693
variable_manager (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase property*), 929
variable_manager (*angr.knowledge_plugins.key_definitions.definition.Definition attribute*), 1245
variable_map (*angr.ailment.Manager attribute*), 212
variable_map (*angr.ailment.manager.Manager attribute*), 224
variable_map_of() (*in module angr.analyses.decompiler.variable_map*), 727
variable_offset() (*angr.analyses.decompiler.variable_map.VariableMap method*), 728
variable_offset (*angr.analyses.decompiler.ssailification.traversal_engine.DefInfo property*), 656
VariableAccessing (*class in angr.knowledge_plugins.variables.variable_access*), 1282
VariableAccess (*class in angr.protos.variables_pb2*), 1405
VariableAccessSort (*class in angr.knowledge_plugins.variables.variable_access*), 1282
VariableAnnotation (*class in angr.analyses.variable_recovery.variable_recovery_base*), 926
VariableBase (*class in angr.protos.variables_pb2*), 1405
VariableInUseEntry (*class in angr.protos.codegen_pb2*), 1401
VariableManager (*class in angr.knowledge_plugins.variables.variable_manager*), 1288
VariableManagerInternal (*class in angr.knowledge_plugins.variables.variable_manager*), 1283
VariableManagerInternal (*class in angr.protos.variables_pb2*), 1405
VariableManagerSerializer (*class in angr.angrdb.serializers.variables*), 955
VariableMatchPredicate (*class in angr.analyses.decompiler.variable_map*), 727
VariableNameSuggestions (*class in angr.llm_models*), 1295
VariableNotFoundError, 437
VariableProxiNode (*class in angr.analyses.proximity_graph*), 799
VariableRecovery (*class in angr.analyses.variable_recovery.variable_recovery*), 926
VariableRecoveryBase (*class in angr.analyses.variable_recovery.variable_recovery_base*), 927
VariableRecoveryFast (*class in angr.analyses.variable_recovery.variable_recovery_fast*), 930
VariableRecoveryFastState (*class in angr.analyses.variable_recovery.variable_recovery_fast*), 930
VariableRecoveryState (*class in angr.analyses.variable_recovery.variable_recovery*), 925
VariableRecoveryStateBase (*class in angr.analyses.variable_recovery.variable_recovery_base*), 927
VariableRename (*class in angr.llm_models*), 1294

VARIABLES (*angr.analyses.complete_calling_conventions.CallingConventionAnalysis attribute*), 353
variables (*angr.analyses.variable_recovery.variable_recovery_base.VariableRecoveryStateBase property*), 929
variables (*angr.state_plugins.sim_action_object.SimActionObject property*), 1586
variables (*angr.storage.memory_object.SimMemoryObject property*), 1667
variables() (*angr.state_plugins.solver.SimSolver method*), 1595
variables_unify() (*angr.analyses.decompiler.structured_codegen.base.BaseStructuredCodeGenerator method*), 686
VariableSourceAnnotation (*class in angr.analyses.variable_recovery.annotations*), 923
VariableType (*class in angr.knowledge_plugins.variables.variable_manager*), 1283
VariableType (*class in angr.protos.variables_pb2*), 1405
VariableTypeChange (*class in angr.llm_models*), 1295
VariableTypeSuggestions (*class in angr.llm_models*), 1295
variance (*angr.analyses.typehoon.typevars.BaseLabel property*), 919
variance (*angr.analyses.typehoon.typevars.Store property*), 919
Variance (*class in angr.analyses.typehoon.variance*), 920
variant (*angr.analyses.decompiler.structured_codegen.rust.RustLet property*), 700
variant() (*angr.analyses.decompiler.variable_map.VariableMap method*), 729
VarNode (*class in angr.analyses.loop_analysis.loop_analysis*), 787
Vault (*class in angr.vaults*), 1704
VaultDict (*class in angr.vaults*), 1704
VaultDir (*class in angr.vaults*), 1704
VaultDirShelf (*class in angr.vaults*), 1705
VaultPickler (*class in angr.vaults*), 1704
VaultShelf (*class in angr.vaults*), 1705
VaultUnpickler (*class in angr.vaults*), 1704
VecOutliner (*class in angr.rust.optimization_passes.outliners.vec_outliner*), 1426
vector_args() (*angr.engines.vex.claripy.irop.SimIROp method*), 1074
vector_begin_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 517
vector_cap_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 517
vector_convert_name() (*in module angr.analyses.decompiler.structured_codegen.base*), 670
vector_end_offset() (*in module angr.analyses.decompiler.known_patterns.layouts*), 517
VendorPreset (*class in angr.misc.plugins*), 1320
Veritesting (*class in angr.analyses.veritesting*), 931
Veritesting (*class in angr.exploration_techniques.veritesting*), 1109
VeritestingError, 931
version (*angr.analyses.decompiler.region_overlay.OverlayManager property*), 624
VERSION (*angr.angrdb.AngrDB attribute*), 938
VERSION (*angr.angrdb.db.AngrDB attribute*), 940
vex (*angr.block.Block property*), 961
VEX_LIFT (*angr.state_plugins.inspect.EventType attribute*), 1565
vex_lift_addr (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
vex_lift_addr (*angr.state_plugins.InspectAttrs attribute*), 1497
vex_lift_buff (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
vex_lift_buff (*angr.state_plugins.InspectAttrs attribute*), 1497
vex_lift_size (*angr.state_plugins.inspect.InspectAttrs attribute*), 1566
vex_lift_size (*angr.state_plugins.InspectAttrs attribute*), 1497
vex_nostmt (*angr.block.Block property*), 961
vex_stmt_idx (*angr.ailment.Manager attribute*), 212
vex_stmt_idx (*angr.ailment.manager.Manager attribute*), 224
VEXCCallExpression (*class in angr.ailment.expression*), 223
VEXEarlyExit, 1078
VEXIRSBScanner (*class in angr.analyses.variable_recovery.irsb_scanner*), 925
VEXLifter (*class in angr.engines.vex.lifter*), 1079
VEXMemVar (*class in angr.analyses.propagator.vex_vars*), 798
VEXMixin (*class in angr.engines.vex.light.light*), 1080
vexop_to_simop() (*in module angr.engines.vex.claripy.irop*), 1074
VEXReg (*class in angr.analyses.propagator.vex_vars*), 798
VEXResilienceMixin (*class in angr.engines.vex.light.resilience*), 1080
VEXSlicingMixin (*class in angr.engines.vex.light.slicing*), 1081
VEXStmtDetails (*class in angr.state_plugins.unicorn_engine*), 1600
VEXTmp (*class in angr.analyses.propagator.vex_vars*), 798

VEXVariable (*class in angr.analyses.propagator.vex_vars*), 798
VFG (*class in angr.analyses.vfg*), 934
VFG_HANDLE_SUCCESSOR (*angr.state_plugins.inspect.EventType attribute*), 1565
VFGJob (*class in angr.analyses.vfg*), 932
VFGNode (*class in angr.analyses.vfg*), 934
view() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 627
view() (*angr.storage.memory_mixins.paged_memory.pages.symbolic_bitmap.SymbolicBitmap method*), 1654
view_graph() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 626
view_with_successors() (*angr.analyses.decompiler.region_overlay.RegionOverlay method*), 627
Virtual (*angr.analyses.disassembly.SootExpressionInvoke attribute*), 741
VirtualAlloc (*class in angr.procedures.win32.VirtualAlloc*), 1383
VirtualProtect (*class in angr.procedures.win32.VirtualProtect*), 1383
VirtualVariable (*class in angr.ailment.expression*), 223
VirtualVariable (*class in angr.knowledge_plugins.key_definitions.atoms*), 1244
visit() (*angr.rust.utils.ail.CallVisitor method*), 1438
visited_blocks (*angr.analyses.reaching_definitions.function_handler.FunctionCallData attribute*), 840
visited_blocks (*angr.analyses.reaching_definitions.FunctionCallData attribute*), 807
visited_blocks (*angr.analyses.reaching_definitions.reaching_definitions.ReachingDefinitionsAnalysis property*), 868
visitor (*angr.analyses.reaching_definitions.subject.Subject property*), 870
VlaDimEntry (*class in angr.protos.codegen_pb2*), 1402
VSA_DDG (*class in angr.analyses.vsa_ddg*), 936
vsnprintf (*class in angr.procedures.libc.vsnprintf*), 1362
Vtable (*angr.knowledge_plugins.cfg.indirect_jump.IndirectJumpType attribute*), 1175
Vtable (*angr.knowledge_plugins.cfg.IndirectJumpType attribute*), 1163
Vtable (*class in angr.analyses.vtable*), 937
VtableFinder (*class in angr.analyses.vtable*), 937
vvar_id_at() (*angr.analyses.decompiler.known_patterns.generator.PatternGenerator method*), 515
vvar_load() (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier method*), 580
vvar_store() (*angr.analyses.decompiler.optimization_passes.inlined_string_transformation_simplifier.InlinedStringTransformationSimplifier method*), 580
VVarAliasVisitor (*class in angr.analyses.decompiler.optimization_passes.static_vvar_rewriter*), 595
VVarAndTmpUsesCollector (*class in angr.utils.ssa.combined_uses_collector*), 1698
VVarRewritingVisitor (*class in angr.analyses.decompiler.optimization_passes.static_vvar_rewriter*), 595
VVarUsesCollector (*class in angr.utils.ssa.vvar_uses_collector*), 1699

## W

walk() (*angr.ailment.block_walker.AILBlockWalker method*), 215
walk() (*angr.analyses.decompiler.ailgraph_walker.AILGraphWalker method*), 398
walk() (*angr.analyses.decompiler.expression_narrower.ExpressionNarrower method*), 452
walk() (*angr.analyses.decompiler.region_walker.RegionWalker method*), 642
walk() (*angr.analyses.decompiler.sequence_walker.SequenceWalker method*), 652
walk_expression() (*angr.ailment.block_walker.AILBlockWalker method*), 216
walk_node() (*angr.analyses.decompiler.optimization_passes.ite_expr_converter.ITEExprConverter method*), 582
walk_node() (*angr.analyses.decompiler.region_walker.RegionWalker method*), 642
walk_statement() (*angr.ailment.block_walker.AILBlockWalker method*), 215
WARNING (*angr.analyses.decompiler.notes.decompilation_note.DecompilationNote attribute*), 538
WARNING (*angr.analyses.decompiler.notes.DecompilationNoteLevel attribute*), 538
was_parameter (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 1244
was_reg (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 1244
was_stack (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 1244
was_tmp (*angr.knowledge_plugins.key_definitions.atoms.VirtualVariable property*), 1244
wcscasecmp (*class in angr.procedures.libc.wchar*), 1362
wcscmp (*class in angr.procedures.libc.wchar*), 1362
wDay (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386
wDayOfWeek (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386
WeakAssignment (*class in angr.ailment.statement*), 226
when (*class in angr.state_plugins.inspect*), 1570
where (*angr.analyses.decompiler.known_patterns.KnownPattern attribute*), 451
where (*angr.analyses.decompiler.known_patterns.pattern.KnownPattern attribute*), 521

where (*angr.analyses.decompiler.KnownPattern attribute*), 379
white (*angr.misc.ansi.BackgroundColor attribute*), 1314
white (*angr.misc.ansi.Color attribute*), 1314
whitelist (*angr.analyses.veritesting.CallTracingFilter attribute*), 931
wHour (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386
widen() (*angr.state_plugins.sim_action_object.SimActionObject method*), 1587
widened_jobs (*angr.analyses.forward_analysis.job_info.JobInfo property*), 764
width (*angr.analyses.data_dep.dep_nodes.MemDepNode property*), 360
width() (*angr.analyses.disassembly.DisassemblyPiece method*), 739
WinStackCanarySimplifier (*class in angr.analyses.decompiler.optimization_passes.win_stack_canary_simplifier*), 598
with_all_edges() (*angr.analyses.decompiler.region_overlay.RegionOverlayGraph method*), 632
with_arch() (*angr.rust.sim_type.EnumVariant method*), 1434
with_arch() (*angr.sim_type.SimType method*), 1460
with_arch() (*angr.sim_type.TypeRef method*), 1461
with_details() (*angr.knowledge_plugins.propagations.propagation_model.PropagationModel method*), 1269
with_evidence() (*angr.analyses.decompiler.known_patterns.context.PatternContext method*), 511
with_type() (*angr.state_plugins.debug_variables.SimDebugVariable method*), 1545
with_type() (*angr.state_plugins.view.SimMemView method*), 1607
wMilliseconds (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386
wMinute (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386
wMonth (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386
word() (*angr.analyses.decompiler.known_patterns.context.PatternContext method*), 488
word_size (*angr.analyses.decompiler.known_patterns.context.PatternContext property*), 489
work() (*angr.analyses.complete_calling_conventions.CompleteCallingConventionsAnalysis method*), 354
Worker (*class in angr.distributed.worker*), 994
WRITE (*angr.analyses.smc.TraceActions attribute*), 891
WRITE (*angr.knowledge_plugins.variables.variable_access.VariableAccessSort attribute*), 1282
Write (*angr.knowledge_plugins.xrefs.xref_types.XRefType attribute*), 1293
Write (*angr.knowledge_plugins.xrefs.XRefType attribute*), 1291
WRITE (*angr.state_plugins.sim_action.SimActionData attribute*), 1585
WRITE (*angr.state_plugins.SimActionData attribute*), 1504
Write (*class in angr.procedures.java_io.write*), 1338
write (*class in angr.procedures.posix.write*), 1376
write() (*angr.storage.file.SimFileBase method*), 1611
write() (*angr.storage.file.SimFileDescriptorBase method*), 1614
write() (*angr.storage.file.SimPackets method*), 1613
write_data() (*angr.storage.file.SimFileDescriptorBase method*), 1614
write_msr() (*angr.state_plugins.unicorn_engine.Unicorn method*), 1604
write_pos (*angr.storage.file.SimFileDescriptor property*), 1616
write_pos (*angr.storage.file.SimFileDescriptorBase property*), 1615
write_pos (*angr.storage.file.SimFileDescriptorDuplex property*), 1616
write_storage (*angr.storage.file.SimFileDescriptor property*), 1616
write_storage (*angr.storage.file.SimFileDescriptorBase property*), 1615
write_storage (*angr.storage.file.SimFileDescriptorDuplex property*), 1616
write_to() (*angr.knowledge_plugins.variables.variable_manager.VariableManagerInternal method*), 1283
WriteFile (*class in angr.procedures.win32.file_handles*), 1384
writev (*class in angr.procedures.linux_kernel.iovec*), 1366
wSecond (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386
wYear (*angr.procedures.win32.sim_time.GetLocalTime attribute*), 1386

## X

X86CCallRewriter (*class in angr.analyses.decompiler.ccall_rewriters.x86_ccalls*), 463
X86ElfPicPltResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.x86_elf_pic_plt*), 341
x86g_calculate_aad_aam() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_calculate_condition() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_calculate_daa_das_aaa_aas() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_calculate_eflags_all() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_calculate_eflags_c() (*in module angr.engines.vex.claripy.ccall*), 1072

x86g_calculate_RCL() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_calculate_RCR() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_check_fldcw() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_create_fpucw() (*in module angr.engines.vex.claripy.ccall*), 1072
x86g_dirtyhelper_CPUID_sse0() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_CPUID_sse2() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_CPUID_sse3() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_FINIT() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_IN() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_LGDT_LIDT() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_loadF80le() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_OUT() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_RDMSR() (*in module angr.engines.vex.heavy.dirty*), 1078
x86g_dirtyhelper_RDTSC() (*in module angr.engines.vex.heavy.dirty*), 1076
x86g_dirtyhelper_storeF80le() (*in module angr.engines.vex.heavy.dirty*), 1078
x86g_dirtyhelper_SxDT() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_write_cr0() (*in module angr.engines.vex.heavy.dirty*), 1077
x86g_dirtyhelper_XGETBV() (*in module angr.engines.vex.heavy.dirty*), 1078
x86g_use_seg_selector() (*in module angr.engines.vex.claripy.ccall*), 1072
X86GccGetPcSimplifier (*class in angr.analyses.decompiler.optimization_passes.x86_gcc_getpc_simplifier*), 598
X86PeIatResolver (*class in angr.analyses.cfg.indirect_jump_resolvers.x86_pe_iat*), 341
xalloc_die (*class in angr.procedures.gnulib.xalloc_die*), 1337
Xor (*angr.engines.light.ArithmeticExpression attribute*), 1006
Xor (*angr.engines.light.data.ArithmeticExpression attribute*), 1010
XRef (*class in angr.knowledge_plugins.xrefs.xref*), 1291
XrefDict (*class in angr.knowledge_plugins.xrefs.xref_manager*), 1292
XRefManager (*class in angr.knowledge_plugins.xrefs.xref_manager*), 1292
xrefs (*angr.knowledge_plugins.functions.function.Function property*), 1212
XRefs (*class in angr.protos.xrefs_pb2*), 1406
XRefsAnalysis (*class in angr.analyses.xrefs*), 938
XRefsSerializer (*class in angr.angrdb.serializers.xrefs*), 956
XRefType (*class in angr.knowledge_plugins.xrefs.xref_types*), 1293
xstrtol_fatal (*class in angr.procedures.gnulib.xstrtol_fatal*), 1337

## Y

yellow (*angr.misc.ansi.BackgroundColor attribute*), 1314
yellow (*angr.misc.ansi.Color attribute*), 1314
Yield (*angr.protos.primitives_pb2.Edge attribute*), 1404

## Z

zen_hook() (*in module angr.state_plugins.trace_additions*), 1597
zen_memory_write() (*in module angr.state_plugins.trace_additions*), 1597
zen_register_write() (*in module angr.state_plugins.trace_additions*), 1597
ZenPlugin (*class in angr.state_plugins.trace_additions*), 1597
zeroextend_on_demand() (*in module angr.utils.bits*), 1674